Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή εργασία

Τίτλος:

Μελέτη και υλοποίηση του αλγορίθμου VQM για εκτίμηση ποιότητας τηλεοπτικής εικόνας

Σαράντου Φώτιος (ΑΜ: 1673)

Επιβλέπων καθηγητής: Γεώργιος Γαρδίκης

Επιτροπή Αξιολόγησης: Γεώργιος Γαρδίκης, Ευάγγελος Πάλλης, Γεώργιος Ξυλούρης

Ημερομηνία παρουσίασης: 13/9/2010

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τους συναδέλφους μου στο ΑΤΕΙ Κρήτης, τους καθηγητές μου και ιδιαίτερα τον επιβλέποντα κ. Γεώργιο Γαρδίκη, επιστημονικό συνεργάτη του ΤΕΙ που με βοήθησε από τεχνικής πλευράς στο να υλοποιήσω τον αλγόριθμο VQM σε γλώσσα προγραμματισμού C++.

Ακόμα ευχαριστώ όλους εκείνους που μοιράζονται πολύτιμες και δωρεάν πληροφορίες στο διαδίκτυο και τέλος τους γονείς μου, που μου συμπαραστάθηκαν σ' αυτή την προσπάθεια.

Σύνοψη

Η ψηφιακή τηλεόραση έγινε πραγματικότητα χάρη στην ανάπτυξη των ασύρματων επικοινωνιών και των αλγορίθμων συμπίεσης. Όμως ακόμα υπάρχει πιθανότητα το βίντεο που θα παρακολουθεί ο τηλεθεατής να έχει υποστεί παραμόρφωση, είτε από λάθη κατά τη μετάδοση, είτε από υπερβολική συμπίεση.

Στην παρούσα διπλωματική εργασία περιγράφεται και αναπτύσσεται σε μορφή λογισμικού ένας προτυποποιημένος αλγόριθμος που μετράει αντικειμενικά την ποιότητα της ψηφιακής τηλεοπτικής εικόνας. Ο αλγόριθμος αυτός χρησιμοποιείται για να προσεγγίσει την υποκειμενική ποιότητα της εικόνας, όπως γίνεται αντιληπτή από τον χρήστη και βοηθάει στην ανίχνευση προβλημάτων στο δίκτυο παροχής της ψηφιακής τηλεόρασης και στη βελτιστοποίησή του. Στα πλαίσια της διπλωματικής αναπτύχθηκε μονάδα λογισμικού που υλοποιεί πλήρως τον αλγόριθμο και δοκιμάστηκε σε ένα σύνολο από βίντεο που είχαν υποστεί κάποιου είδους παραμόρφωση.

Προκειμένου να είναι ακριβής και κοντά στην πραγματικότητα, πρέπει η βαθμολόγηση της ποιότητας να γίνεται με βάση το βίντεο που παρακολουθεί ο τηλεθεατής και όχι το σήμα που λαμβάνεται από την κεραία, επειδή η πραγματικά αντιληπτή ποιότητα της εικόνας δεν εξαρτάται μόνο από τις παραμέτρους και τη συμπεριφορά του δικτύου αλλά και από δυναμικά χαρακτηριστικά του αρχικού βίντεο όπως χωρική ανάλυση, κίνηση και δυναμική της εικόνας.

Abstract

Digital Television became reality thanks to the development of wireless communications and compression algorithms. However there is still possibility that the video watched by the viewer is deformed by errors in transmission or excessive compression.

This thesis presents the architecture and the software development of a standardized software-based algorithm which measures objectively the quality of digital television image. This algorithm is used to approach the picture quality, as perceived by the user and can assist in detecting problems in the video delivery chain and in optimizing it. In the frame of the thesis, a software module fully implementing the algorithm is developed and it is tested against a set of impaired videos.

In order to be accurate and as close to reality as possible, the assessment of quality must be based on the video watched by the viewer and not on the quality of the signal received by the antenna, because the actual perceived quality does not only depend on network parameters but also on dynamic characteristics of the video such as spatial analysis, motion and picture dynamics.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Λίστα Πινάκων

1 Ψηφιακή τηλεόραση

1.1 Περιγραφή

Η ψηφιακή τηλεόραση είναι ένα σύστημα μετάδοσης ήχου και εικόνας σε ψηφιακή μορφή σε αντιδιαστολή με την υπάρχουσα αναλογική τηλεόραση. Επιτυχώς αντικαθιστά προοδευτικά το αναλογικό σύστημα σε όλο τον κόσμο.

Υπάρχουν πολλοί τρόποι για την λήψη ψηφιακής τηλεόρασης. Ο συνηθέστερος χρησιμοποιεί μια κεραία επίγειου σήματος (επίγεια ψηφιακή τηλεόραση DVB-T). Άλλα δημοφιλή συστήματα είναι η ψηφιακή καλωδιακή τηλεόραση DVB-C και η δορυφορική ψηφιακή τηλεόραση DVB-S. Υπάρχει ακόμα ένα πρότυπο, το DVB-H για την λήψη ψηφιακού σήματος από κινητές συσκευές.

Σύγχρονα ψηφιακά συστήματα δίνουν στους τηλεθεατές τη δυνατότητα της αλληλεπίδρασης με προγράμματα μέσω διαδραστικών εφαρμογών όπως τηλεπαιχνίδια, τηλεψηφοφορίες, τηλεαγορές. Για να επιτευχθεί αυτό απαιτείται κανάλι επιστροφής που να στέλνει δεδομένα πίσω στον τηλεοπτικό σταθμό [1].

1.2 Τεχνικές πληροφορίες

Η ψηφιακή τηλεόραση υποστηρίζει πολλά διαφορετικά πρότυπα εικόνας που ορίζονται από τον συνδυασμό του μεγέθους, το aspect ratio (λόγος πλάτος προς ύψος), interlacing (πεπλεγμένη σάρωση). Πεπλεγμένη σάρωση είναι η μέθοδος μετάδοσης μόνο των μισών γραμμών σε κάθε καρέ. Μεταδίδονται δηλαδή εναλλάξ μόνο οι μονές ή μόνο οι ζυγές γραμμές. Το μάτι ελάχιστα αντιλαμβάνεται την διαφορά. Ως προς την ανάλυση που χρησιμοποιείται, η ψηφιακή εκπομπή χωρίζεται σε δύο γενικές κατηγορίες, στην HDTV και SDTV.

Τα χαρακτηριστικά της HDTV είναι 1280 x 720 pixels σε προοδευτική σάρωση ή 1920 x 1080 σε πεπλεγμένη σάρωση [2].

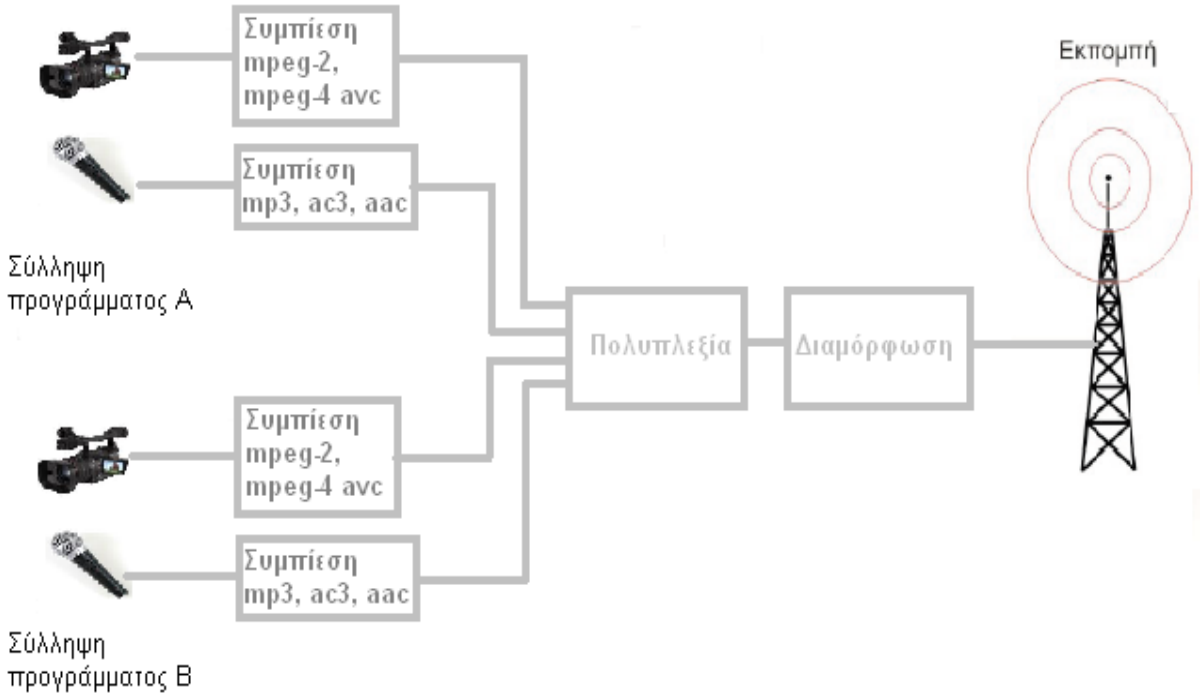
Στην SDTV ανάλογα με τη χώρα και το aspect ratio χρησιμοποιούνται τρεις αναλύσεις. Για 4:3 εκπομπές, 640 x 480 χρησιμοποιείται στις χώρες που χρησιμοποιούν το αναλογικό πρότυπο NTSC και 720 x 576 στις χώρες που χρησιμοποιούν το αναλογικό πρότυπο PAL. Για 16:9 εκπομπές, 704 x 480 στις χώρες με NTSC και 720 x 576 στις χώρες με PAL.

1.3 Αλυσίδα ψηφιακής τηλεόρασης

Στον ψηφιακό τηλεοπτικό πομπό εκτελούνται οι εξής διαδικασίες:

1. Κωδικοποίηση/Συμπίεση: Το ψηφιακό βίντεο συμπιέζεται με τεχνικές επεξεργασίας εικόνας. Παρόμοια διαδικασία γίνεται για την μουσική και την ομιλία με τεχνικές επεξεργασίας ήχου.
2. Πολυπλεξία: Είναι η διαδικασία του συνδυασμού της πληροφορίας πολλών τηλεοπτικών προγραμμάτων σε ένα κανάλι. Έτσι δεν χρειάζεται ξεχωριστό κανάλι για κάθε πρόγραμμα.
3. Διαμόρφωση/Εκπομπή: Για την προστασία και τη μετάδοση του σήματος χρησιμοποιούνται από το διαμορφωτή τρεις βασικοί μηχανισμοί, η κωδικοποίηση καναλιού, η διεμπλοκή και η διαμόρφωση.

Παρακάτω περιγράφονται αναλυτικά αυτές οι διαδικασίες.



Εικόνα 1: Αλυσίδα ψηφιακής τηλεόρασης.

1.3.1 Κωδικοποίηση/Συμπίεση

Ένα αναλογικό σήμα PAL καταλαμβάνει φάσμα 8MHz. Αν ψηφιοποιηθεί, το φάσμα που απαιτείται, εκτιμάται στα 120 MHz. Με τεχνικές ψηφιακής επεξεργασίας εικόνας μειώνεται το απαιτούμενο φάσμα κατά 97%. Αποτέλεσμα είναι αντί για 248 Mbit/sec (για HDTV, 1 Gbit/sec) απαιτούνται μόνο 6 Mbit/sec για τη μετάδοση ενός ψηφιοποιημένου σήματος PAL. Υπάρχουν δύο πρότυπα που χρησιμοποιούνται στην ψηφιακή τηλεόραση, το MPEG-2 και το MPEG-4. Στη συνέχεια περιγράφεται το MPEG-2 που χρησιμοποιείται και στα DVD.

Οι τεχνικές που χρησιμοποιούνται για τη συμπίεση είναι η αντιστάθμιση κίνησης, μετασχηματισμός συνημιτόνου, κωδικοποίηση εντροπίας. Αρχικά η εικόνα μετατρέπεται από RGB σε YUV, χωρίζεται σε blocks των 8x8 pixels και σε macroblocks των 16x16 pixels.

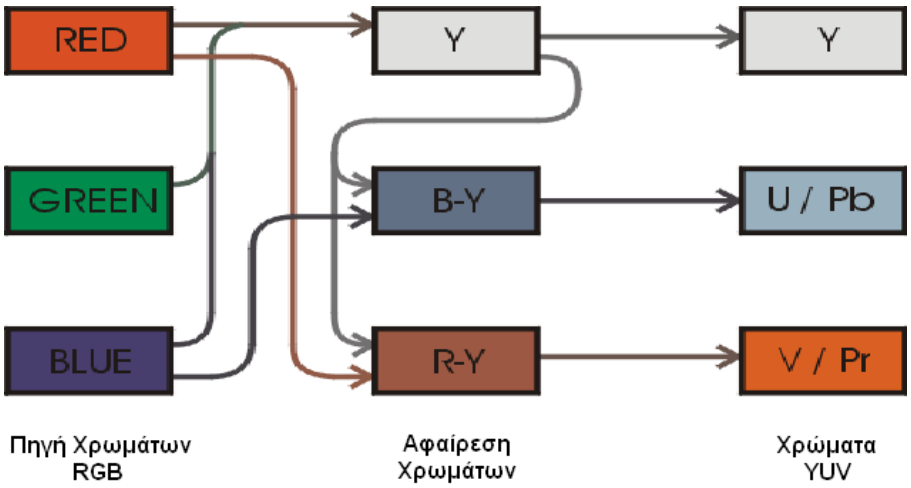
Για λόγους συμβατότητας αλλά και εξοικονόμησης φάσματος, αντί για το σύστημα χρωμάτων RGB χρησιμοποιείται το YUV όπου Y είναι η φωτεινότητα (όπως στην ασπρόμαυρη τηλεόραση), U είναι η χρωματικότητα του μπλε (αλλιώς Cb) και V είναι η χρωματικότητα του κόκκινου (αλλιώς Cr).

Ο μετασχηματισμός από το σύστημα RGB στο YUV γίνεται ως εξής:

$Y = 0.299R + 0.587G + 0.114B$

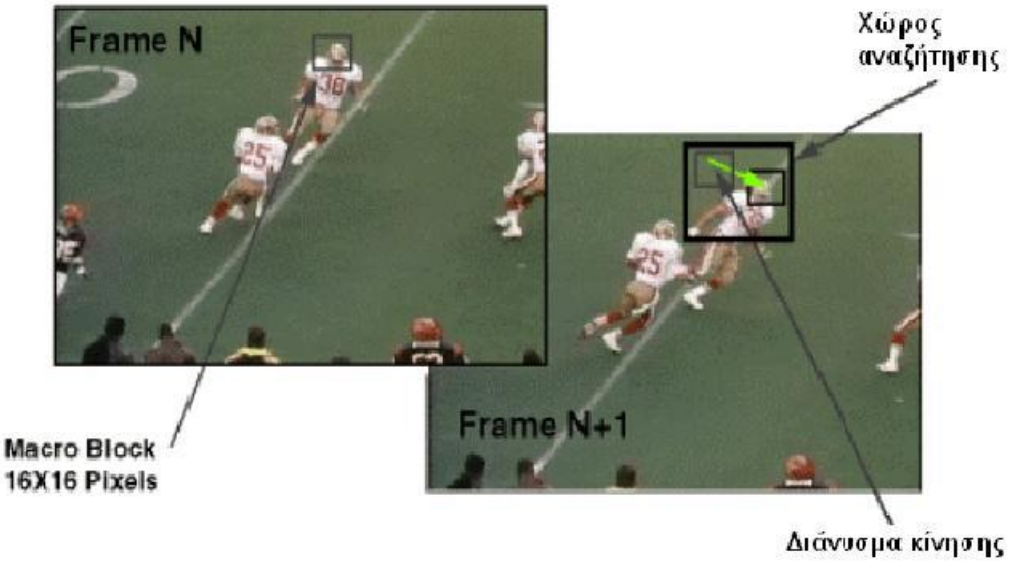
$U = 0.492(B - Y)$

$V = 0.877(R - Y)$



Εικόνα 2: Μετατροπή από RGB σε YUV.

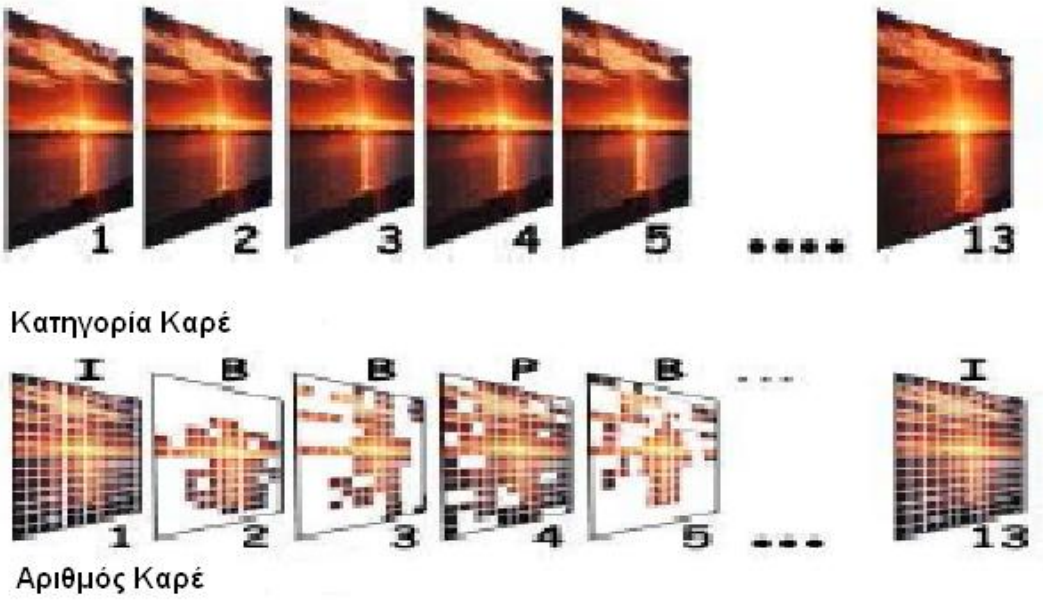
Στην αντιστάθμιση κίνησης εκμεταλλευόμαστε την ομοιότητα ανάμεσα σε διαδοχικά καρέ. Σε κάθε καρέ αναζητούμε macroblocks που είναι παρόμοια με macroblocks γειτονικών καρέ που είτε μετακινήθηκαν είτε έμειναν στάσιμα. Αποθηκεύεται μόνο το διάνυσμα κίνησης του macroblock από το ένα καρέ στο άλλο, αντί για το ίδιο το περιεχόμενο του macroblock και έτσι μεταδίδονται μόνο τα διανύσματα κίνησης και όχι τα macroblocks. Αν δεν βρεθεί κατάλληλο ταίριασμα, το macroblock θεωρείται αυτόνομο και κωδικοποιείται εξολοκλήρου χωρίς αναφορά [3].



Εικόνα 3: Αντιστάθμιση κίνησης.

Τα καρέ ταξινομούνται σε τρεις κατηγορίες, ως εξής:

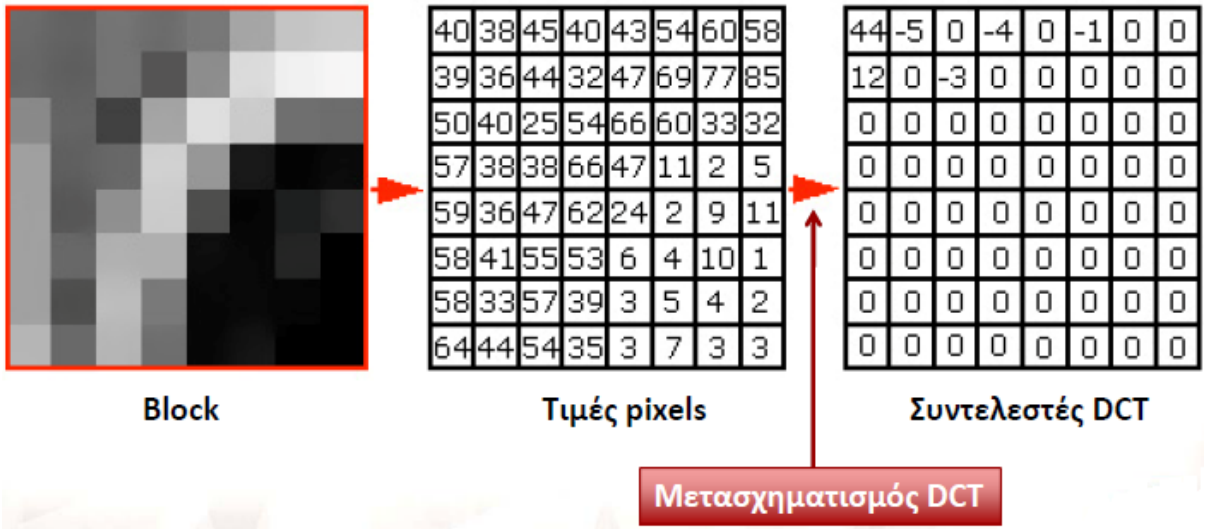
1. I-frames (Intra): Είναι καρέ που κωδικοποιούνται ολόκληρα και δεν εξαρτώνται από τα γειτονικά τους.
2. P-frames (Predicted): Είναι καρέ που κωδικοποιούνται με βάση τα προηγούμενα I-frames και P-frames. Απαιτούν περίπου το 1/3 των δεδομένων των I-frames [4].
3. B-frames (Bidirectional): Είναι καρέ που κωδικοποιούνται με βάση τα προηγούμενα και τα επόμενα I-frames και P-frames.



Εικόνα 4: Ταξινόμηση καρέ.

Για να μην συνεχίζουν ενδεχόμενα λάθη να μεταδίδονται από καρέ σε καρέ, κάθε τόσο πρέπει να μεταδίδονται ανεξάρτητα καρέ.

Στον μετασχηματισμό διακριτού συνημιτόνου DCT οι τιμές των pixels κάθε block 8x8 αντιστοιχούνται σε συντελεστές χωρικών συχνοτήτων. Ο μετασχηματισμός DCT είναι μία απωλεστική μέθοδος συμπίεσης. Επιλέγουμε και κωδικοποιούμε τους συντελεστές του μετασχηματισμού οι οποίοι περιέχουν την μεγαλύτερη ενέργεια, και απορρίπτουμε τους υπόλοιπους. Η εικόνα ανακτάται (με απώλειες) με την αντιστροφή του μετασχηματισμού.

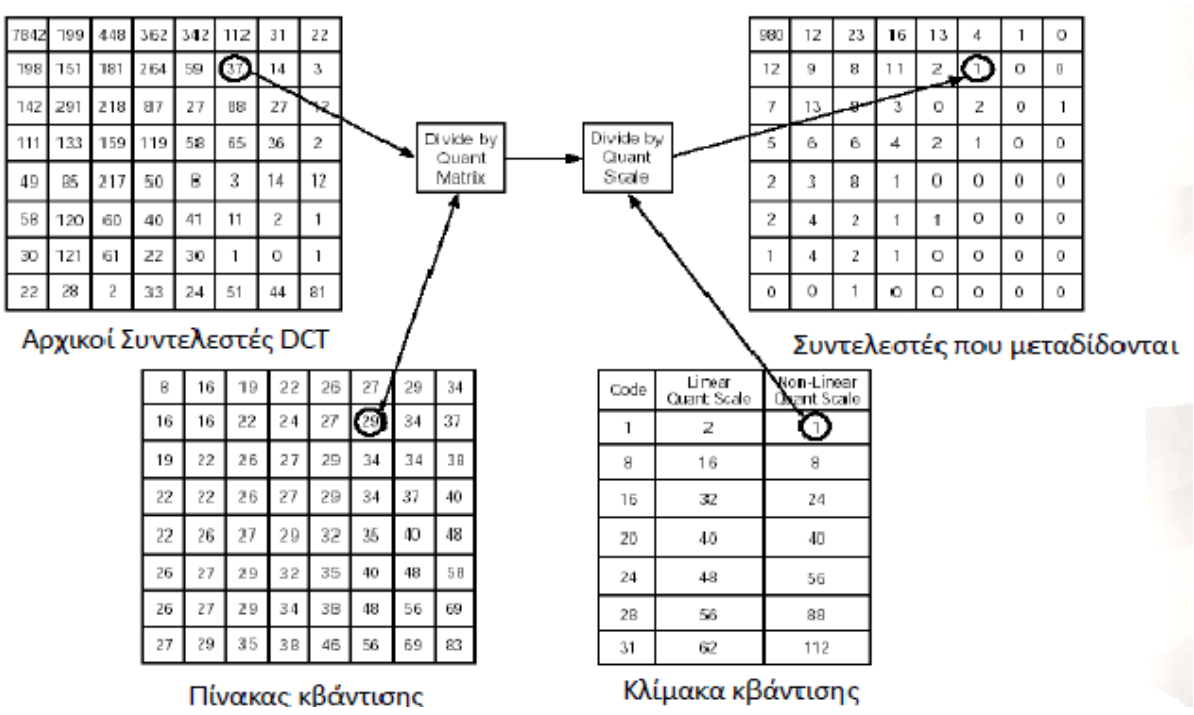


Εικόνα 5: Μετασχηματισμός DCT.

Για μεγαλύτερη εξοικονόμηση bits, οι συντελεστές DCT δεν μεταδίδονται ως έχουν, αλλά κβαντίζονται. Συγκεκριμένα, γίνονται τα εξής:

1. Κάθε συντελεστής διαιρείται με το αντίστοιχο στοιχείο ενός πίνακα κβάντισης (Quantizer Matrix) που είναι σταθερός και προκαθορισμένος .

2. Το αποτέλεσμα διαιρείται με έναν σταθερό αριθμό που λέγεται κλίμακα κβάντισης (Quantizer Scale)ο οποίος επιλέγεται κατά την κωδικοποίηση.
3. Μεταδίδεται τελικά το τελικό πηλίκο των δύο διαδοχικών διαιρέσεων (η στρογγυλοποίηση του σε ακέραιο).



Εικόνα 6: Μετασχηματισμός DCT – κβάντιση.

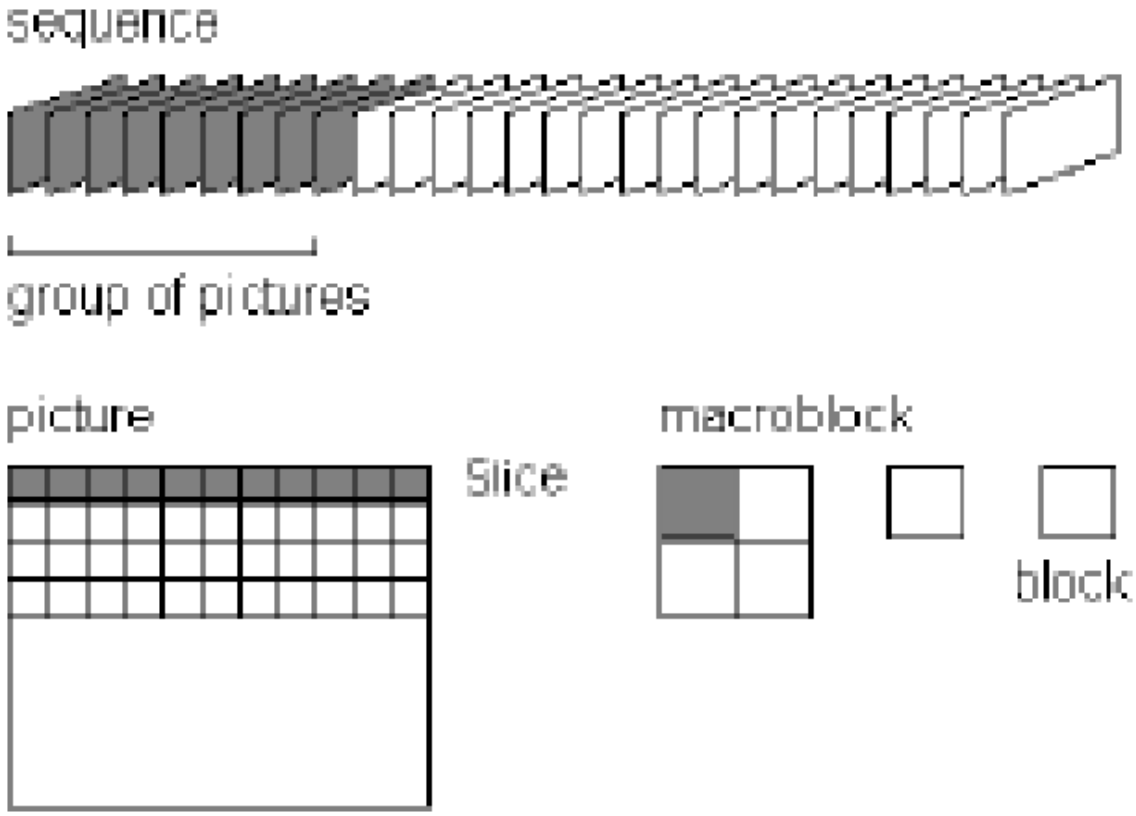
Όσο αυξάνουμε την κλίμακα κβάντισης, τόσο περισσότεροι είναι οι συντελεστές DCT που τελικά μηδενίζονται και δεν μεταδίδονται. Έτσι, εξοικονομούμε bits αλλά χάνουμε σε ποιότητα εικόνας καθώς χάνονται χωρικές συχνότητες. Τα block παραμορφώνονται, χάνουν τη λεπτομέρεια τους αλλά και τη συνέχειά τους. Γίνονται δηλαδή ορατά τα όρια των blocks, κάτι που είναι γνωστό ως blockiness effect.

Εικόνα 7: Παράδειγμα συμπίεσης.

Στην κωδικοποίηση εντροπίας (Huffman), παράγεται ένας κώδικας βασισμένος στην πιθανότητα εμφάνισης του κάθε συντελεστή σε ένα βίντεο. Σχεδόν σε όλα τα βίντεο, μερικοί συντελεστές εμφανίζονται περισσότερες φορές από ότι άλλοι. Προκαθορισμένες πιθανότητες εμφάνισης κάθε συντελεστή χρησιμοποιούνται για τη δημιουργία ενός δυαδικού δέντρου από τη βάση προς τα επάνω (bottom-up). Αυτός ο τρόπος εγγυάται ότι οι συντελεστές που εμφανίζονται λιγότερο θα έχουν μακρύτερες σειρές δυαδικών ψηφίων. Στο δέντρο οι συντελεστές είναι φύλλα (τερματικοί κόμβοι - terminal nodes), οι διακλαδώσεις σημειώνονται με 0 ή 1 και η δυαδική αναπαράσταση της διαδρομής από τη ρίζα (root) μέχρι τον συντελεστή είναι η συμπιεσμένη αναπαράστασή του ως σειρά δυαδικών ψηφίων.

1.3.2 Πολυπλεξία

Το μικρότερο δομικό στοιχείο είναι το block 8x8 με τους συντελεστές DCT είτε για τη φωτεινότητα ή τη χρωματικότητα. Τέσσερα blocks φωτεινότητας (και συνήθως ένα χρωματικότητας μπλε και ένα χρωματικότητας κόκκινου) σχηματίζουν ένα macroblock. Μια σειρά από macroblocks σχηματίζει ένα slice. Το σύνολο των slices στην εικόνα σχηματίζει ένα καρέ (frame). Μια ομάδα από καρέ σχηματίζει ένα GOP (Group of Pictures). Το σύνολο των GOP σχηματίζει το Video Sequence. Η δομή που προαναφέρθηκε αποτελεί ουσιαστικά την έξοδο του κωδικοποιητή εικόνας. Αποτελεί ένα συνεχές bitstream που ονομάζεται Στοιχειώδης Ροή (Elementary Stream – ES).



Εικόνα 8: Δομή του Elementary Stream.

Το Elementary Stream δεν είναι κατάλληλο για μετάδοση, επειδή:

1. Δεν έχει διαιρεθεί σε πακέτα.
2. Δεν περιέχει ήχο.

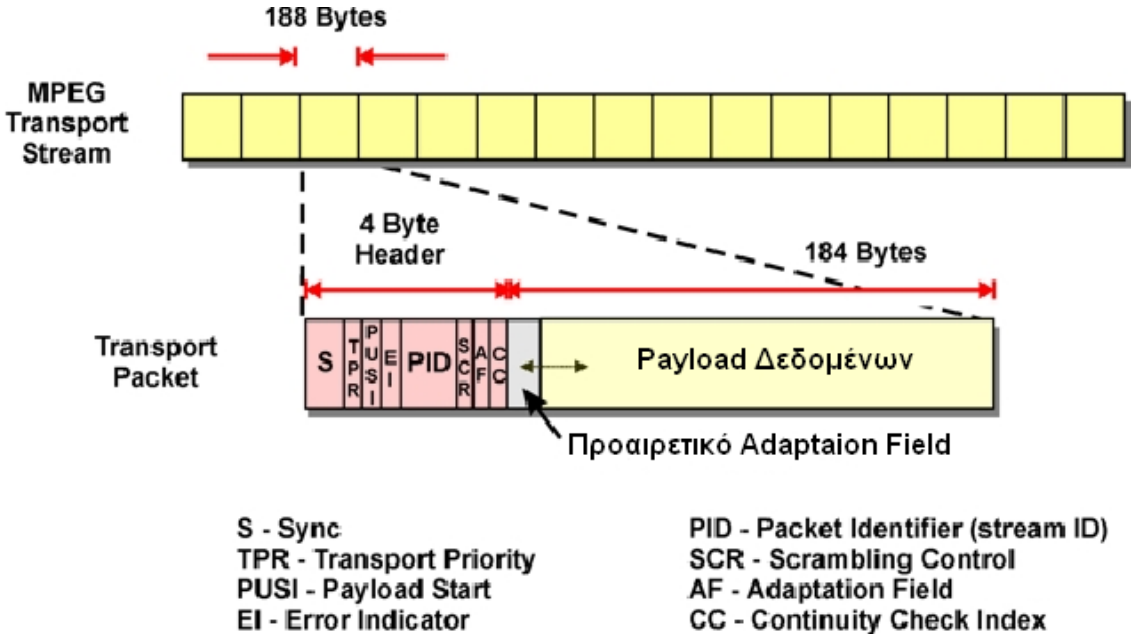
3. Δεν περιέχει πληροφορίες προγράμματος.
4. Δεν μπορεί να πολυπλεχθεί με άλλα προγράμματα.
5. Έχει συγκεκριμένες αδυναμίες απέναντι σε σφάλματα μετάδοσης.

Το πρώτο βήμα είναι η διαίρεση του Elementary Stream σε πακέτα. Προκύπτει το Packetised Elementary Stream (PES) και τα πακέτα ονομάζονται PES packets. Συνήθως ο διαχωρισμός γίνεται στα όρια των καρέ και κάθε καρέ αντιστοιχεί σε ένα PES packet. Συνεπάγεται ότι τα PES packets δεν έχουν σταθερό μήκος. Με τον ίδιο τρόπο σχηματίζεται και ένα PES για τον κωδικοποιημένο ήχο.

Στο κάθε PES packet επισυνάπτεται μία επικεφαλίδα (PES packet header). Μεταξύ άλλων, η επικεφαλίδα περιέχει το μήκος του πακέτου, το αναγνωριστικό του Elementary Stream που μεταφέρει, καθώς και πληροφορίες χρονισμού (PTS/DTS) που ενημερώνουν τον αποκωδικοποιητή πότε να αποκωδικοποιήσει και πότε να παρουσιάσει το περιεχόμενο του πακέτου.

Το Packetised Elementary Stream είναι χωρισμένο σε πακέτα. Όμως τα PES packets είναι μεταβλητού μεγέθους, είναι πολύ μεγάλα και δεν είναι κατάλληλα ούτε για μετάδοση, ούτε για πολυπλεξία. Για τον λόγο αυτό, γίνεται μια δεύτερη διαίρεση σε πακέτα. Το Packetised Elementary Stream διαιρείται και αυτό σε πακέτα σταθερού μεγέθους των 188 bytes που λέγονται Πακέτα Μεταφοράς (Transport Packets Tps).

Το κάθε Πακέτο Μεταφοράς έχει σταθερό μήκος 188 bytes και επικεφαλίδα 4 bytes. Άρα το ωφέλιμο φορτίο του (payload) είναι 184 bytes, όπου περιέχονται bytes από το PES. Το πεδίο προσαρμογής (adaptation field) έχει κενά bytes και χρησιμοποιείται για να γεμίσει το πακέτο όταν τα δεδομένα είναι λιγότερα από 184 bytes. Αυτό συμβαίνει στο τέλος του πακέτου PES.

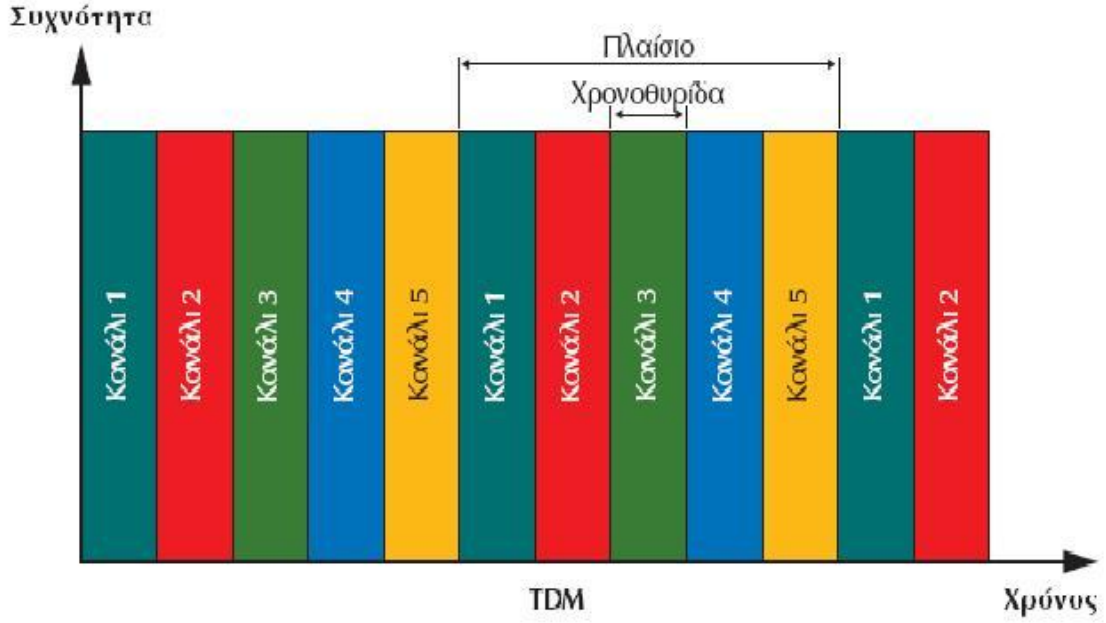


Εικόνα 9: Δομή πακέτου μεταφοράς Transport Stream.

Στην πολυπλεξία συνδυάζονται πληροφορίες πολλών τηλεοπτικών προγραμμάτων σε μία ροή.

Για να γίνει η πολυπλεξία, οι κωδικοποιημένες ροές εικόνας, οι κωδικοποιημένες ροές ήχου, οι πληροφορίες υπηρεσιών και άλλα δεδομένα (διαδραστικές υπηρεσίες, δεδομένα IP κλπ) θα πρέπει να έχουν φορτωθεί σε Transport Packets.

Η πολυπλεξία MPEG 2 είναι τύπου διαίρεσης χρόνου (Time Division Multiplexing – TDM). Η πολυπλεξία διαίρεσης χρόνου είναι τεχνολογία ψηφιακής μετάδοσης σημάτων και χρησιμοποιείται κυρίως στην επικοινωνία ηλεκτρολογικών υπολογιστών. Ο χρόνος διαιρείται σε χρονοθυρίδες (timeslots) και η μεταφορά των πακέτων γίνεται κυκλικά. Δηλαδή δεν εκπέμπεται κάθε πρόγραμμα σε διαφορετική συχνότητα (όπως στην πολυπλεξία διαίρεσης συχνότητας FDM) αλλά τα Transport Packets των διάφορων ροών συνδυάζονται και μεταδίδονται διαδοχικά. Το αποτέλεσμα είναι ένα σύνθετο MPEG 2 Transport Stream (Multi Program Transport Stream, MPTS) που αποτελεί και το τελικό σήμα βασικής ζώνης της ψηφιακής τηλεόρασης. Προκύπτει ότι το συνολικό bitrate της πολυπλεξίας ισούται με το άθροισμα των bitrate των ροών εισόδου. Το συνηθέστερο όμως είναι ο πολυπλέκτης να λειτουργεί σε ένα σταθερό bitrate εξόδου που είναι μεγαλύτερο ή ίσο του αθροίσματος, ακόμη και αν οι ροές εισόδου έχουν μεταβλητό bitrate. Στην περίπτωση αυτή εισάγονται από τον πολυπλέκτη κενά (stuffing) Transport Packets για να αναπληρώσουν την διαφορά [5].



Εικόνα 10: Πολυπλεξία διαίρεσης χρόνου.

1.3.3 Διαμόρφωση/Εκπομπή

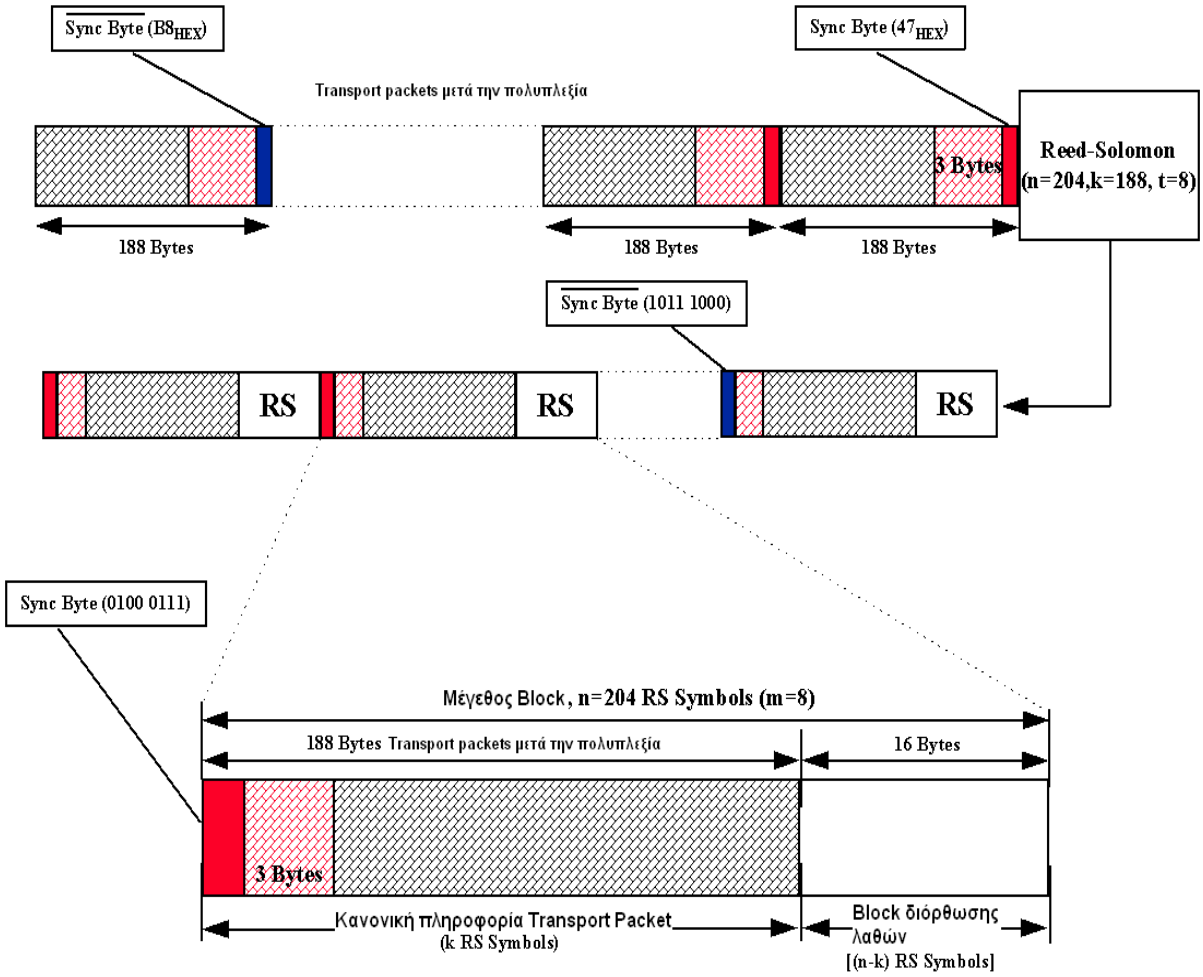
Για την προστασία και τη μετάδοση του σήματος χρησιμοποιούνται από το διαμορφωτή οι τρεις βασικοί μηχανισμοί:

1. Κωδικοποίηση καναλιού (Channel coding).
2. Διεμπλοκή (Interleaving).
3. Διαμόρφωση (Modulation).

Κωδικοποίηση καναλιού (channel coding) ονομάζεται η διαδικασία προσθήκης bytes πλεονασμού στο stream που μεταδίδεται. Αυτά τα επιπλέον bytes δίνουν στον δέκτη τη δυνατότητα να ανιχνεύει και να διορθώνει τα λάθη που συμβαίνουν κατά τη μετάδοση (μέχρι ένα συγκεκριμένο ποσοστό). Όσο περισσότερα bytes προστίθενται, δηλαδή όσο μικρότερο είναι το κλάσμα του ρυθμού κώδικα, τόσο αυξάνεται η ικανότητα διόρθωσης λαθών στο

δέκτη, αλλά μειώνεται το ωφέλιμο διαθέσιμο εύρος ζώνης. Στο DVB-T και στο DVB-S χρησιμοποιούνται για την κωδικοποίηση οι αλγόριθμοι ReedSolomon και συνελικτικός.

Οι κώδικες ReedSolomon είναι μη δυαδικοί κώδικες διόρθωσης λαθών που μπορούν να εντοπίζουν και να διορθώνουν πολλά τυχαία λάθη στα σύμβολα. Προσθέτοντας i ελέγχους συμβόλων στα δεδομένα, ένας RS κώδικας μπορεί να εντοπίσει κάθε συνδυασμό μέχρι i λάθη και να διορθώσει μέχρι t=i/2 λάθη. Ακόμα μπορούν να διορθώσουν συνεχόμενα λάθη. Στην ψηφιακή τηλεόραση ο ReedSolomon προσθέτει σε κάθε πακέτο μεταφοράς των 188 bytes (k=188), 16 bytes πλεονασμού (N=204) και μπορεί να διορθώσει μέχρι 8 λάθη (t=8) [6].



Εικόνα 11: ReedSolomon

Ο συνελικτικός κώδικας είναι παρόμοιος με τον ReedSolomon, δηλαδή προστίθενται bytes πλεονασμού για την διόρθωση λαθών αλλά το πλήθος τους δεν είναι συγκεκριμένο. Όσα περισσότερα bytes προστίθενται, τόσο αυξάνεται η ικανότητα διόρθωσης. Η ικανότητα αυτή ρυθμίζεται με τον ρυθμό κώδικα που είναι ο λόγος των bytes εισόδου στον κωδικοποιητή προς τα bytes εξόδου.

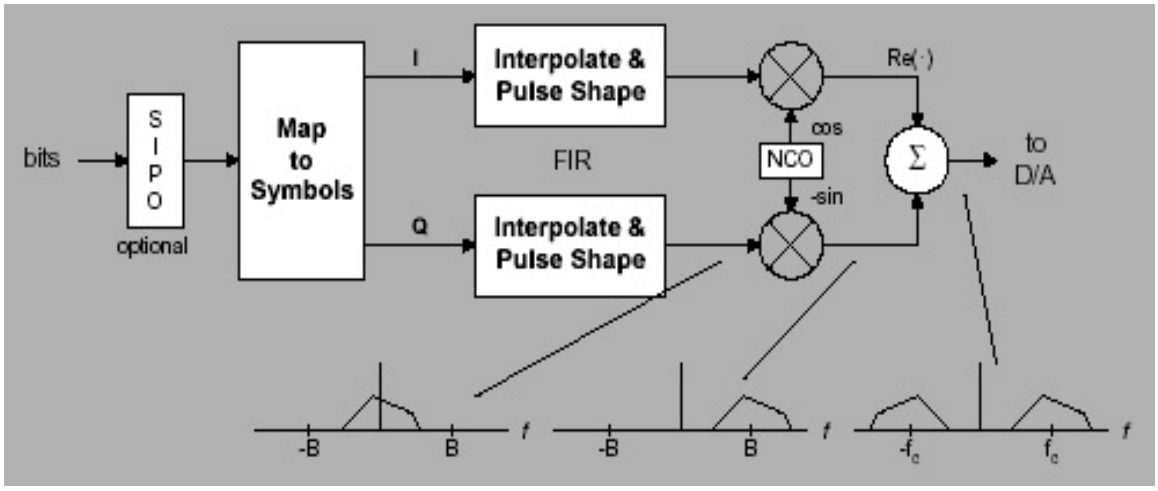
Η διεμπλοκή ανακατεύει (permutes) σύμβολα σύμφωνα με ένα αλγόριθμο απεικόνισης (mapping). Ο αντίστοιχος deinterleaver χρησιμοποιεί την αντίστροφη αντιστοίχιση για να ανακατασκευάσει την αρχική σειρά των συμβόλων. Η διαδικασία του interleaving και του deinterleaving βρίσκουν χρησιμότητα στο να αντιμετωπίζουν τα λάθη ριπής (burst errors) σε ένα τηλεπικοινωνιακό σύστημα.

Στην επίγεια ψηφιακή τηλεόραση χρησιμοποιείται διαμόρφωση πλάτους φάσης (Amplitude Phase Modulation), όπου τόσο το πλάτος όσο και η φάση του φέροντος διαμορφώνονται από την ομάδα από bits (σύμβολο) που εκπέμπεται.

Η λειτουργία της QAM διαμόρφωσης είναι γενικά κάτι αρκετά απλό. Η φιλοσοφία είναι η εξής, διαμορφώνεται μία ημιτονοειδής φέρουσα σε πλάτος και φάση. Κάθε σύμβολο (symbol) είναι ένας συγκεκριμένος συνδυασμός τιμής πλάτους και φάσης.

Το εύρος συχνοτήτων ενός QAM σήματος εξαρτάται από την ροή των δεδομένων (symbol rate). Για παράδειγμα μία ροή δεδομένων της τάξης των 3200 symbols/sec απαιτεί περίπου 3,2KHz εύρος συχνοτήτων. Το πρωτόκολλο V.34, το γνωστό σε όλα τα τηλεφωνικα modems κάνει χρήση της κεντρικής φέρουσας με συχνότητα 1959 Hz και απαιτεί ένα εύρος 3430Hz για να επιτύχει την ταχύτητα μεταφοράς 3430 symbols/sec. Έτσι το εύρος συχνοτήτων που κάνει χρήση είναι από 244KHz μέχρι και 3674KHz.

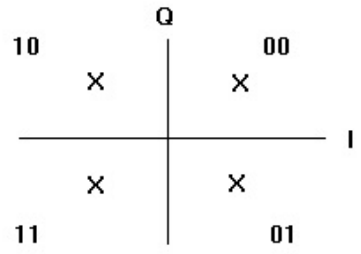
Στην ψηφιακή τηλεόραση η QAM διαμόρφωση αναλύεται ως εξής. Ο διαμορφωτής δέχεται μία ροή δεδομένων και το διαιρεί σε 2 μισής ταχύτητας ροές οι οποίες ονομάζονται I και Q. Η ροή I είναι ένα κατά πλάτος διαμορφωμένο σήμα πάνω στη φέρουσα αναφοράς ενώ η ροή Q είναι και αυτό ένα κατά πλάτος διαμορφωμένο σήμα πάνω στην ίδια φέρουσα με διαφορά φάσης 90 μοιρών. Για το λόγο αυτό η ροή I χαρακτηρίζεται «ως σε φάση» (In-Phase) και η ροή Q ως τετραγωνική συνιστώσα (Quadrature component). Οι δύο αυτές συνιστώσες I και Q έρχονται σε μίξη (συνήθως από ένα διαχωριστή αντεστραμμένο).



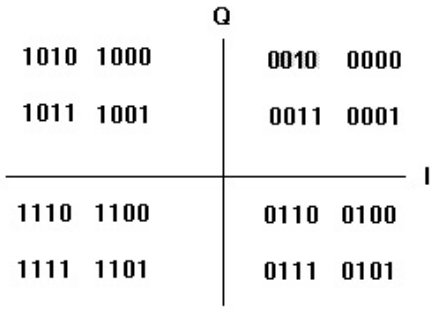
Εικόνα 12: Κύκλωμα QAM.

Λόγω της μεγάλης απόδοσης η διαμόρφωση QAM είναι πολύ διαδεδομένη στην ψηφιακή επικοινωνία. Έχει θεωρηθεί ως standard τεχνολογία για την ενσύρματη και ασύρματη μετάδοση εικόνας , ήχου , φωνής και δεδομένων. Όπως σε κάθε διαμόρφωση έτσι και στην QAM υπάρχουν αρκετά επίπεδα εκπομπής. Κάθε n-bit (σύμβολο) τοποθετείται επάνω σε ένα 2^n κομμάτι ενός 2 διαστάσεων σήματος εκπομπής ο οποίος αναφέρεται ως 2^n – QAM διαμόρφωση. Κάθε σύμβολο λοιπόν έχει x και y συντεταγμένες. Αν λοιπόν θέλουμε να επιτύχουμε το πρώτο επίπεδο εκπομπής 2^2 – QAM (4-QAM) τότε οι ροές δεδομένων I και Q διαμορφώνονται ως ένα απλό σήμα θετικό και αρνητικό μίας φέρουσας.

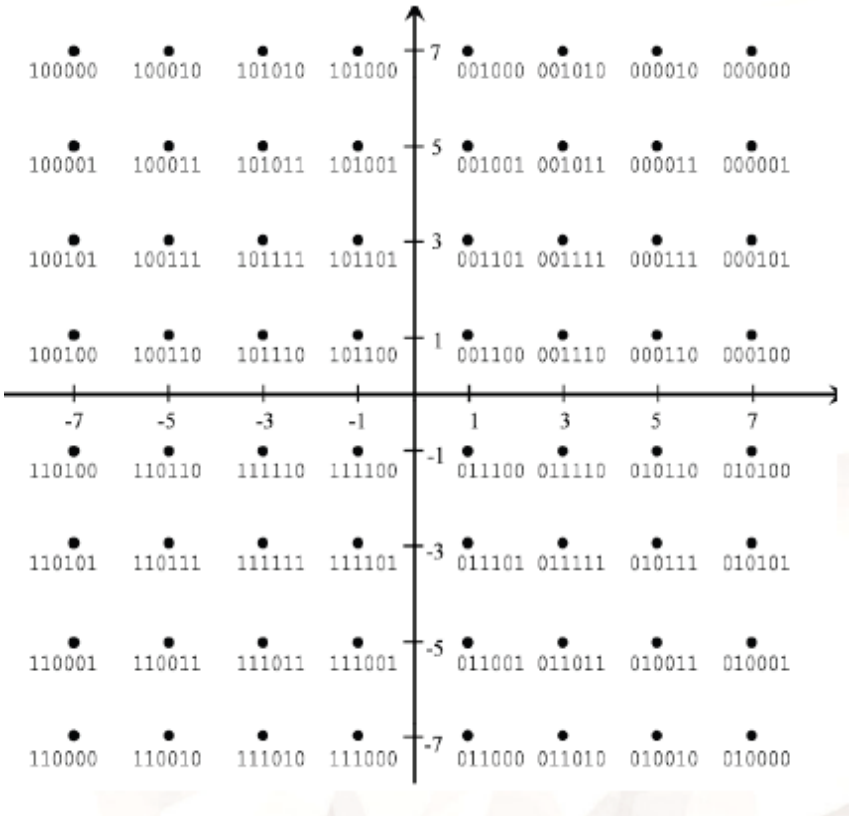
Στην επίγεια ψηφιακή τηλεόραση χρησιμοποιούνται τέσσερις διαμορφώσεις διαφορετικής «τάξης»: QPSK, 16QAM, 64QAM, 256QAM.



Εικόνα 13: QPSK (2 bits/symbol).



Εικόνα 14: 16QAM (4 bits/symbol).



Εικόνα 15: 64QAM (6 bits/symbol).

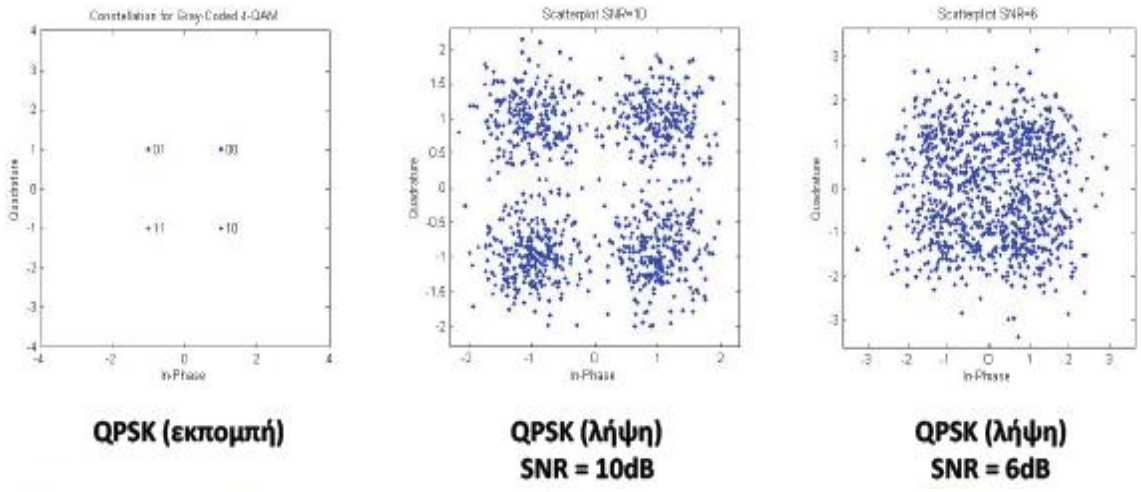
Ο θόρυβος επιδρά αρνητικά στις διαμορφώσεις κατά πλάτος και φάση μετακινώντας την φέρουσα από την πραγματική της θέσης. Υπάρχει ένα ανώτατο όριο ανοχής παρουσίας θορύβου για κάθε σημείο της φέρουσας. Θεωρείται ότι η φέρουσα περιέχει περισσότερο θόρυβο, όταν η κάθε διαμόρφωσή της θα έχει απόκλιση από την πραγματική έως ότου τελικά

ο δέκτης ανακριβώς να ερμηνεύσει λανθασμένα το σύμβολο, για παράδειγμα το 0110 ως 0100.

Υπάρχουν κάποιοι τρόποι για να αντιμετωπιστεί το φαινόμενο αυτό. Ένας από αυτούς είναι να περιοριστούν οι συνεχείς μεταπηδήσεις από μία διαμόρφωση της φέρουσας σε μία άλλη χρησιμοποιώντας διαφορετική μέθοδο κωδικοποίησης. Παραδείγματος χάριν στο ανωτέρω παράδειγμα, εάν η τελευταία τιμή της φέρουσας ήταν 1010, να έχουμε έναν κανόνα ότι η μόνη πιθανή επόμενη αξία στο αμέσως χαμηλότερο τεταρτημόριο θα μπορούσε να είναι 0110. Έτσι ο δέκτης θα έβλεπε 0100 και θα το διόρθωνε σε 0110. Μπορεί αυτό να φαίνεται ότι έτσι γυρνάμε πίσω από την διαμόρφωση 16-QAM στην διαμόρφωση 4-QAM, όμως στην πραγματικότητα περιορίζει την ροή δεδομένων μόνο κατά 33% αν εφαρμοστεί ο κατάλληλος κώδικας.

Οι περισσότεροι κατασκευαστές βέβαια κάνουν χρήση πιο περίπλοκων διαδικασιών διόρθωσης λαθών που επιφέρουν ακόμα μικρότερη μείωση στην πραγματική ροή πληροφορίας. Συμπερασματικά να αναφέρουμε ότι τα μικρότερα επίπεδα διαμόρφωσης QAM μεταφέρουν λιγότερη πληροφορία αλλά είναι και λιγότερο ευάλωτα στον θόρυβο.

Η μέτρηση που έχει καθιερωθεί για το επίπεδο θορύβου στην διαμόρφωση QAM δεν είναι το γνωστό BER (Bit Error Rate) αλλά το MER (Modulation Error Rate). Η μέτρηση είναι απαραίτητη για μία σωστή λήψη ενώ η τιμή της έχει σχέση με τον τύπο της διαμόρφωσης που μετρά (16, 32, 64 QAM…). Οι τιμές κυμαίνονται από 35db για μία πολύ ποιοτική λήψη, έως και 23 db για μία οριακή.



Εικόνα 16: Αποκλίσεις στα σημεία του σηματαστερισμού.

Όσο μεγαλύτερη είναι η τάξη της διαμόρφωσης, τόσο αυξάνεται ο αριθμός των bits που αντιπροσωπεύει κάθε σύμβολο. Επειδή ο αριθμός των συμβόλων που εκπέμπονται ανά δευτερόλεπτο είναι σταθερός, αυξάνεται το bitrate που μπορεί να σταλεί.

Όμως όσο μεγαλύτερη είναι η τάξη της διαμόρφωσης, τόσο πιο κοντά μεταξύ τους είναι τα σημεία του σηματαστερισμού. Έτσι, υπάρχει μεγαλύτερος κίνδυνος σε περίπτωση χαμηλού SNR ένα σύμβολο να ληφθεί λανθασμένα ως κάποιο γειτονικό του. Έτσι, συμβαίνει λάθος το οποίο καλείται να διορθώσει η κωδικοποίηση καναλιού. Αυτό σημαίνει ότι σήμα που εκπέμπεται με διαμόρφωση υψηλότερης τάξης, απαιτεί υψηλότερο SNR στο δέκτη για να ληφθεί σωστά.

Συνεπώς υπάρχουν δύο παράμετροι σε ένα σύστημα επίγειας ψηφιακής τηλεόρασης που ορίζουν τη χωρητικότητα σε ωφέλιμο bitrate και την αντοχή του σήματος:

1. Ρυθμός κώδικα στην κωδικοποίηση καναλιού:
- Με χαμηλό ρυθμό κώδικα, δηλαδή πολλά bits πλεονασμού, το σήμα μπορεί να ληφθεί σωστά ακόμη και στη περίπτωση χαμηλού SNR.
- Με υψηλό ρυθμό κώδικα, το ωφέλιμο bitrate είναι μεγαλύτερο.
2. Τάξη διαμόρφωσης:
- Με διαμόρφωση μικρής τάξης (πχ. QPSK), το σήμα μπορεί να ληφθεί σωστά ακόμη και σε χαμηλό SNR.
- Με διαμόρφωση μεγάλης τάξης (πχ. 256QAM), το ωφέλιμο bitrate είναι μεγαλύτερο.

Ο συνδυασμός σχήματος διαμόρφωσης και ρυθμού κώδικα ρυθμίζεται στον πομπό και λέγεται σχήμα μετάδοσης.

Modulation	Code Rate	Ωφέλιμο bitrate (Mbits/s)	Ελάχιστο απαιτούμενο SNR στον δέκτη (dB)
QPSK	1/2	5,53	3,1
QPSK	2/3	7,37	4,9
QPSK	3/4	8,29	5,9
QPSK	5/6	9,22	6,9
QPSK	7/8	9,68	7,7
16-QAM	1/2	11,06	8,8
16-QAM	2/3	14,75	11,1
16-QAM	3/4	16,59	12,5
16-QAM	5/6	18,43	13,5
16-QAM	7/8	19,35	13,9
64-QAM	1/2	16,59	14,4
64-QAM	2/3	22,12	16,5
64-QAM	3/4	24,88	18,0
64-QAM	5/6	27,65	19,3
64-QAM	7/8	29,03	20,1

Τιμές για σήμα DVB-T σε κανάλι 8MHz, Guard Interval = 1/8

Εικόνα 17: Σχήμα μετάδοσης.

Από το σχήμα μετάδοσης ορίζεται:

1. Το ωφέλιμο bitrate.
2. Το ελάχιστο απαιτούμενο SNR στον δέκτη ώστε να ληφθεί σωστά το σήμα χωρίς να γίνει υπέρβαση της μέγιστης ικανότητας διόρθωσης του αποκωδικοποιητή καναλιού.

Στην επίγεια ψηφιακή λήψη εμφανίζεται έντονα το φαινόμενο της πολυδιαδρομικής διάδοσης (multipath) δηλαδή την λήψη από τον δέκτη πολλαπλών αντιγράφων του σήματος λόγω ανακλάσεων. Η πολυδιαδρομική διάδοση προκαλεί διαφορετική εξασθένηση σε διαφορετικές συχνότητες (συχνό επιλεκτικές διαλείψεις), κάτι που οδηγεί σε παραμόρφωση του φάσματος του σήματος. Για να αντιμετωπιστεί το φαινόμενο αυτό χρησιμοποιείται στην επίγεια ψηφιακή τηλεόραση η τεχνική της Ορθογωνικής Πολυπλεξίας Διαίρεσης Συχνότητας.

Εικόνα 18: Φαινόμενο της πολυδιαδρομικής διάδοσης.

Η Ορθογωνική Πολύπλεξη με Διαίρεση Συχνότητας (Orthogonal Frequency Division Multiplexing, OFDM) είναι μια τεχνική μετάδοσης που επιτρέπει σε ψηφιακά δεδομένα να μεταδοθούν επαρκώς και με αξιοπιστία σε ένα κανάλι επικοινωνίας, ακόμα και αν αυτό περιλαμβάνει πολυδιαδρομική διάδοση. Με την τεχνική OFDM τα δεδομένα μεταδίδονται χρησιμοποιώντας ένα μεγάλο αριθμό καναλιών μικρού εύρους ζώνης. Τα κανάλια αυτά χωρίζονται σε τακτά διαστήματα συχνότητας σχηματίζοντας το φάσμα του μεταδιδόμενου σήματος. Η συχνοτική απόσταση των φερουσών επιλέγεται κατά τέτοιο τρόπο ώστε να είναι ορθογώνιες μεταξύ τους και να μην παρεμβάλεται το περιεχόμενο της μιας στην άλλη. Αυτό επιτυγχάνεται παρ' όλη τη συχνοτική επικάλυψη της μιας φέρουσας με την άλλη [7].

Το OFDM επιτρέπει ακόμη τη χρήση μονοσυχνοτικών δικτύων (Single Frequency Networks – SFNs) δηλαδή την κάλυψη μιας μεγάλης περιοχής από πολλές διεσπαρμένες κεραίες εκπομπής που λειτουργούν στην ίδια συχνότητα και εκπέμπουν το ίδιο ακριβώς ψηφιακό τηλεοπτικό σήμα. Αυτό δεν ήταν δυνατόν στην αναλογική τηλεόραση, όπου κεραίες γειτονικών περιοχών έπρεπε να λειτουργούν σε διαφορετικές συχνότητες για την αποφυγή παρεμβολών, ακόμη και αν εξέπεμπαν το ίδιο σήμα.

1.4 Πλεονεκτήματα της ψηφιακής τηλεόρασης

1. Η ποιότητα της εικόνας είναι σταθερή.
2. Υποστηρίζει πολυκάναλο ήχο και εικόνα υψηλής ευκρίνειας.
3. Η ρύθμιση της κεραίας είναι απλούστερη ακόμη και σε λήψη εσωτερικού χώρου.
4. Περισσότερα προγράμματα μπορούν να μεταδοθούν στο ίδιο φάσμα.
5. Υποστηρίζει διαδραστικές υπηρεσίες.
6. Ευκολότερη αποθήκευση και επεξεργασία του περιεχομένου.

1.5 Μειονεκτήματα της ψηφιακής τηλεόρασης

1. Απαιτείται καινούργιος εξοπλισμός
2. Ελαφρώς αυξημένη κατανάλωση ενέργειας (στην περίπτωση ξεχωριστής τηλεοπτικής συσκευής και ψηφιακού δέκτη)
3. Στην αναλογική τηλεόραση όταν λαμβάνεται σήμα με χαμηλό σηματοθορυβικό λόγο η εικόνα ανανεώνεται αλλά έχει παράσιτα ενώ στην ψηφιακή τηλεόραση η εικόνα παγώνει και σταματάει να ανανεώνεται.
4. Η αλλαγή των προγραμμάτων είναι πιο αργή λόγω των καθυστερήσεων στην επεξεργασία της πολυπλεξίας,

Παρά τις προαναφερθείσες τεχνικές που χρησιμοποιούνται για την προστασία του σήματος, σε συνθήκες πολύ κακής λήψης είναι φυσικό να παρουσιάζεται απώλεια δεδομένων στον δέκτη. Η απώλεια αυτή προκαλεί παραμορφώσεις στην ψηφιακή εικόνα, που μπορεί να είναι ιδιαίτερα έντονες και ενοχλητικές στον χρήστη-τηλεθεατή. Οι επόμενες ενότητες παρουσιάζουν αλγορίθμους που βασίζονται σε τεχνικές επεξεργασίας εικόνας και χρησιμοποιούνται για την αποτίμηση της ποιότητας μετά από τυχόν απώλειες.

2 Εκτίμηση ποιότητας τηλεοπτικής εικόνας

2.1 Γιατί χρειάζεται

Παραμόρφωση στην τηλεοπτική εικόνα μπορεί να συμβεί είτε κατά το στάδιο της συμπίεσης, είτε κατά το στάδιο της εκπομπής.

Στο στάδιο της συμπίεσης όπως αναφέρθηκε στο προηγούμενο κεφάλαιο χρησιμοποιούνται τεχνικές σαν τον μετασχηματισμό DCT που μειώνει το απαιτούμενο bitrate αλλά προσθέτει παραμόρφωση στο αρχικό βίντεο και σε περιπτώσεις που ο κωδικοποιητής δε ρυθμιστεί σωστά εμφανίζονται φαινόμενα όπως φαινόμενο blockiness.

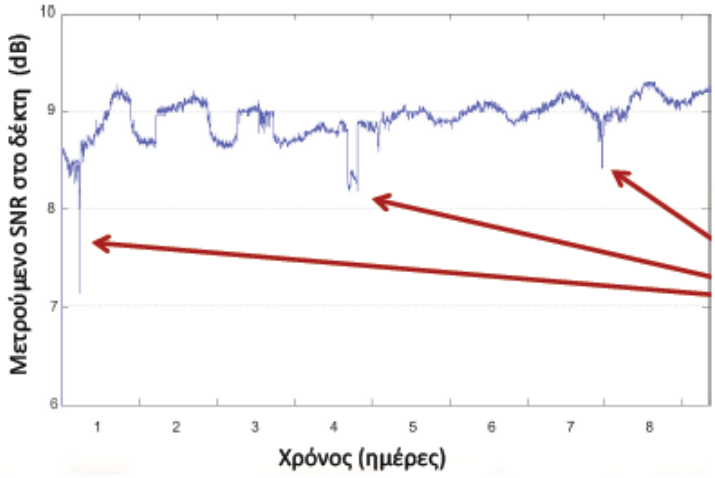
Στο στάδιο της εκπομπής, ανάλογα με το σύστημα μετάδοσης που χρησιμοποιείται, εμφανίζονται κάποια προβλήματα που παρεμβάλλουν και αλλοιώνουν το αρχικό σήμα.

Στην περίπτωση του επίγειου συστήματος ψηφιακής τηλεόρασης, εμφανίζονται φαινόμενα όπως:

1. Απώλειες του σήματος λόγω σκίασης, δηλαδή της παρεμβολής εμποδίων μεταξύ του πομπού και του δέκτη, όπως κτίρια, δέντρα, βουνά κλπ.
2. Η πολυδιαδρομική διάδοση, που οδηγεί στη λήψη από τον δέκτη πολλαπλών αντιγράφων του σήματος από διαφορετικές ανακλάσεις και προκύπτει παραμόρφωση.
3. Όταν κινείται ο δέκτης, εμφανίζεται ολίσθηση Doppler στο πεδίο της συχνότητας.
4. Όταν το περιβάλλον διάδοσης είναι πολύπλοκο, οδηγεί σε απότομες μεταβολές του σηματοθορυβικού λόγου (SNR).
5. Παρεμβολές από ηλεκτρικές συσκευές.

Στην περίπτωση του δορυφορικού συστήματος ψηφιακής τηλεόρασης εμφανίζονται φαινόμενα όπως:

1. Απώλειες του σήματος λόγω μεγάλης απόστασης.
2. Διαλείψεις και εξασθένηση του σήματος λόγω καιρικών φαινομένων όπως βροχή, καταιγίδες κλπ.
3. Ατέλειες στον δορυφορικό αναμεταδότη, επειδή υπάρχει απαίτηση για μικρό μέγεθος και κυρίως για χαμηλή κατανάλωση ενέργειας.
4. Παρεμβολές από γειτονικούς δορυφόρους που εκπέμπουν στο ίδιο κανάλι.
5. Παρεμβολές από πηγές ηλεκτρομαγνητικού θορύβου, όπως ήλιος, ηλεκτρικές εκκενώσεις στην ατμόσφαιρα.



Εικόνα 19: Πτώσεις στο SNR λόγω βροχόπτωσης.

2.2 Λειτουργία αλγορίθμων

Από τότε που έγινε η πρώτη καταγραφή βίντεο, σχεδιάστηκαν πολλά συστήματα επεξεργασίας βίντεο. Στις εποχές του αναλογικού βίντεο, η εκτίμηση ποιότητας ενός συστήματος επεξεργασίας βίντεο γινόταν με τη μέτρηση της απόκρισης συχνότητας χρησιμοποιώντας κλασικές μεθόδους εξέτασης του σήματος (πχ. μπάρες χρωμάτων και κύκλοι). Τώρα τα ψηφιακά συστήματα αντικαθιστούν τα αναλογικά και άλλαξαν οι μέθοδοι εκτίμησης ποιότητας [8].

Δύο τεχνικές υπάρχουν, οι υποκειμενικές και οι αντικειμενικές. Στις υποκειμενικές εκπαιδευμένα άτομα εξετάζουν την εικόνα και αποφασίζουν για την ποιότητά της.

Οι αντικειμενικές τεχνικές για εκτίμηση ποιότητας τηλεοπτικής εικόνας γίνεται με μαθηματικά μοντέλα που προσεγγίζουν την υποκειμενική εκτίμηση αλλά βασίζονται σε αυτοματοποιημένες μετρήσεις που εκτελούνται από υπολογιστή.

Οι αντικειμενικές τεχνικές κατηγοριοποιούνται βάση της ύπαρξης source video που θεωρείται υψηλής ποιότητας σε Full Reference(FR), Reduced Reference(RR) και No Reference(NR). Η FR μέθοδος μετράει την διαφορά ποιότητας συγκρίνοντας κάθε pixel του αρχικού βίντεο με κάθε pixel του παραμορφωμένου. Η RR μέθοδος εξάγει μερικές ιδιότητες από τα δύο βίντεο και τις συγκρίνει για να υπολογίσει το τελικό αποτέλεσμα και χρησιμοποιείται σε περιπτώσεις όπου δεν έχουμε ολοκληρωτική πρόσβαση στο αρχικό βίντεο, πχ. όταν στέλνεται από χαμηλής ταχύτητας μέσου μετάδοσης. Η NR μέθοδος προσπαθεί να αναγνωρίσει την ποιότητα του παραμορφωμένου βίντεο μη έχοντας πρόσβαση στο αρχικό.

Στις επόμενες ενότητες περιγράφονται και υλοποιούνται δύο αλγόριθμοι (metrics) εκτίμησης ποιότητας βίντεο: Το metric PSNR (Peak Signal-to-Noise Ratio) και το metric VQM (Video Quality Measurement).

3 Αλγόριθμος PSNR

3.1 Περιγραφή

Η φράση peak signal-to-noise ratio, σαν συντομογραφία αναφέρεται PSNR. Το PSNR είναι ένας επιστημονικός όρος για την αναλογία της μέγιστης δυνατής ισχύος του σήματος και της ισχύος του θορύβου που έχει προστεθεί και έχει παραμορφώσει το αρχικό βίντεο. Επειδή πολλά σήματα έχουν μεγάλο δυναμικό εύρος, το PSNR εκφράζεται σε λογαριθμική κλίμακα [9].

Το PSNR χρησιμοποιείται συνήθως για την μέτρηση της ποιότητας ανακατασκευασμένων βίντεο ύστερα από την εφαρμογή αλγορίθμων συμπίεσης. Το σήμα σ' αυτήν την περίπτωση είναι το αρχικό βίντεο και ο θόρυβος από λάθη που έχουν προστεθεί κατά τη συμπίεση. Όταν συγκρίνονται αλγόριθμοι συμπίεσης χρησιμοποιείται για την προσέγγιση της ανθρώπινης εκτίμησης, αν και σε μερικές περιπτώσεις ένα βίντεο μπορεί να μοιάζει περισσότερο στο αρχικό, ενώ το PSNR να το έχει βαθμολογήσει χειρότερα από ένα άλλο που μοιάζει λιγότερο στο αρχικό βίντεο. Από αυτή την άποψη, τα αποτελέσματα του αλγορίθμου PSNR μπορεί να είναι ιδιαίτερα ανακριβή.

Το PSNR είναι αλγόριθμος πλήρους αναφοράς (full-reference) που σημαίνει ότι απαιτείται η παρουσία και του αρχικού και του τελικού (παραμορφωμένου) βίντεο. Επίσης, η εφαρμογή του αλγορίθμου PSNR προϋποθέτει ότι οι δοκιμές αναφέρονται στο ίδιο περιεχόμενο και οι κωδικοποιήσεις έγιναν με τον ίδιο αλγόριθμο συμπίεσης.

Το PSNR υπολογίζεται μόνο για την φωτεινότητα επειδή η ανθρώπινη όραση είναι πιο ευαίσθητη σε διαφορές της ποσότητας του φωτός. Έτσι, μετρώντας μόνο τη φωτεινότητα δεν λαμβάνει υπόψιν τις απώλειες από τη μετατροπή του συστήματος RGB σε YUV.

Ο τύπος του PSNR είναι:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

όπου:

- $MaxErr^2$ είναι η μέγιστη διαφορά ενέργειας που μπορούν να έχουν τα Y σήματα των δύο βίντεο.
- MSE είναι η μέση διαφορά ενέργειας που έχουν τα Y σήματα των δύο βίντεο (αρχικού και τελικού).
- Το PSNR είναι ο λόγος των δύο παραπάνω σε λογαριθμική κλίμακα decibel.
- Το $MaxErr^2$ για 8 bit components είναι $255^2 = 65025$.

Ένα περαιτέρω κριτήριο εκτιμητή είναι το μέσο τετραγωνικό σφάλμα (mean squared error) MSE. Αυτό χρησιμοποιείται κυρίως για μεροληπτικούς εκτιμητές. Για αμερόληπτους το μέσο σφάλμα (bias) ισούται με μηδέν, οπότε το μέσο τετραγωνικό σφάλμα είναι ίσο με τη διασπορά του εκτιμητή. Επομένως για έναν αμερόληπτο εκτιμητή η ελαχιστοποίηση του μέσου τετραγωνικού σφάλματος ταυτίζεται με την αποτελεσματικότητα.

MSE είναι η παραμόρφωση που υφίσταται η εικόνα μετά την ανασύστασή της στο δέκτη και τον αντίστροφο μετασχηματισμό.

Το MSE δίνεται από τον τύπο:

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

όπου για ένα καρέ το:

- m είναι το πλάτος της εικόνας.
- n το ύψος της εικόνας.
- με I συμβολίζεται το αρχικό βίντεο.
- με K το παραμορφωμένο.
- i είναι η στήλη στην οποία βρίσκεται το pixel που εξετάζεται.
- j είναι η γραμμή στην οποία βρίσκεται το pixel που εξετάζεται.

Όταν συγκρίνουμε ένα αρχικό βίντεο με ένα βίντεο που έχει υποστεί συμπίεση το PSNR συνήθως κυμαίνεται από 30 έως 50 db. Για ένα βίντεο που έχει υποστεί συμπίεση και μετάδοση, ένα PSNR από 20 έως 25 db θεωρείται αρκετό. Γενικά ισχύει, όσο μεγαλύτερο PSNR τόσο λιγότερη η παραμόρφωση και αντίστροφα. Όταν δύο βίντεο είναι όμοια, το MSE θα ισούται με μηδέν και θα οδηγήσει σε άπειρο PSNR.

Η απόδοση ενός αλγορίθμου δεν μπορεί να κρίνεται μόνο από το PSNR, γιατί είναι μία έκφραση του τετραγώνου του σφάλματος, αλλά και από την υποκειμενική αντίληψη της ποιότητας της αναπαραγόμενης εικόνας (perceptual criterion).

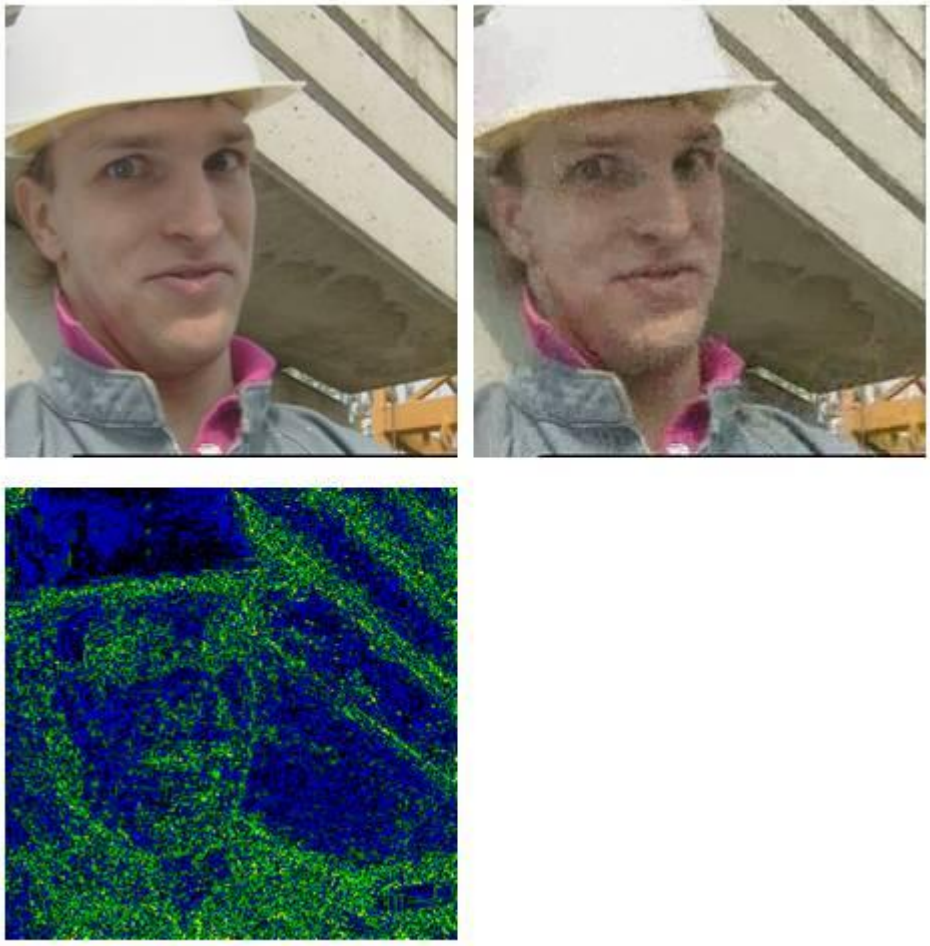
Αν και το PSNR είναι ο πιο παραδοσιακός και πιο συνηθισμένος τρόπος για την μέτρηση εκτίμησης ποιότητας που παράγει ένα σύστημα επεξεργασίας εικόνας, αρκετά πιο προχωρημένοι και πιο ακριβείς αλγόριθμοι έχουν αναπτυχθεί όπως οι:

- Ο structural similarity SSIM εκτελεί τρεις λειτουργίες και εμφανίζει ένα τελικό αποτέλεσμα. Βρίσκει διαφορές στην φωτεινότητα, στην αντίθεση και στην δομικότητα και τις συνδυάζει.
- Ο αλγόριθμος CZD είναι ένας αλγόριθμος σαν το PSNR και μετράει διαφορές μεταξύ των pixel.
- Ο αλγόριθμος VQM που περιγράφεται αναλυτικά στο επόμενο κεφάλαιο προσεγγίζει σε μεγάλο βαθμό την ανθρώπινη όραση.

Όλοι οι παραπάνω αλγόριθμοι έχουν τυποποιηθεί από την ITU International Telecommunication Union.

3.2 Παράδειγμα υπολογισμού PSNR

Στο παραμορφωμένο βίντεο έχει προστεθεί θόρυβος. Παρακάτω βλέπουμε πως αντιλαμβάνεται ο αλγόριθμος PSNR την παραμόρφωση. Με πράσινο παριστάνονται τα σημεία που υπάρχουν μεγάλες διαφορές, με μπλε τα σημεία που υπάρχουν μικρές διαφορές και με μαύρο είναι τα σημεία που δεν υπήρξαν αλλαγές [10].



Εικόνα 20: Αναπαράσταση PSNR

4 Αλγόριθμος VQM

4.1 Εισαγωγή

Ο αλγόριθμος VQM που υλοποιήθηκε στην παρούσα εργασία έχει αναπτυχθεί στην υπηρεσία NTIA (National Telecommunications and Information Administration) των ΗΠΑ και έχει προτυποποιηθεί από την ITU ως ο ακριβέστερος από τους αλγορίθμους εκτίμησης υποκειμενικής ποιότητας εικόνας. Ο αλγόριθμος VQM μπορεί να υλοποιηθεί και ως Reduced Reference. Στην παρούσα εργασία έχει υλοποιηθεί ως Full Reference.

Ο VQM αντιλαμβάνεται τυχόν προβλήματα που προέκυψαν κατά την μετάδοση (παραμορφώσεις, απώλειες, blockiness, πάγωμα εικόνας) και μιμούμενος τη συμπεριφορά της ανθρώπινης όρασης υπολογίζει το πόσο ορατά είναι τα προβλήματα αυτά στο ανθρώπινο μάτι.

Αυτό το αυτόματο σύστημα μέτρησης ποιότητας προσφέρει αποτελέσματα που μοιάζουν με τις βαθμολογίες που έχουν δώσει κατά μέσο όρο οι τηλεθεατές για διάφορα σενάρια.

Η απόδοση ενός συστήματος μέτρησης εκτίμησης ποιότητας γίνεται με τη σύγκριση των αντικειμενικών αποτελεσμάτων σε σχέση με τα υποκειμενικά αποτελέσματα. Το τελευταίο ονομάζεται μέσο ρεκόρ απόψεων (mean opinion score) MOS.

4.2 Υλοποίηση

Πρέπει να σημειωθεί ότι το VQM είναι ένα «αρνητικό» metric, δηλ. όσο μεγαλύτερη είναι η τιμή του, τόσο πιο έντονες φαίνονται οι παραμορφώσεις του βίντεο σε μια υποκειμενική εκτίμηση. Το τελικό αποτέλεσμα του VQM προκύπτει από το άθροισμα με βάρη (weighted sum) έξι μετρικών (metrics) ως εξής:

1. +0.0192*color_coher_color_8x8_1Fmean_euclid_std_10%
2. +0.0431*Y_contrast_ati_4x4_6F_std_ratio_gain_mean_10%
3. -0.2097*Y_si13_8x8_6F_std_12_ratio_loss_below5%
4. +0.5969*Y_hv13_angle0.225_rmin20_8x8_6F_mean_3_ratio_loss_below5%_mean_square
5. +0.2483*Y_hv13_angle0.225_rmin20_8x8_6F_mean_3_log_gain_above95%_mean
6. +2.3416*Y_si13_8x8_6F_std_8_log_gain_mean

Η σημασία και ο τρόπος υπολογισμού των μετρικών αναλύεται στις επόμενες ενότητες.

4.2.1 Χρωματικές απώλειες

Ο παράγοντας αυτός εκφράζει απώλειες στην χρωματική πληροφορία. Τέτοιες απώλειες δημιουργούνται κατά κανόνα από την λανθασμένη ρύθμιση του κωδικοποιητή εικόνας.

Οι χρωματικές απώλειες στο άθροισμα του VQM score εκφράζονται από τον όρο: +0.0192*color_coher_color_8x8_1Fmean_euclid_std_10%

Εκτελούνται τα εξής:

1. Τα βίντεο (αρχικό και τελικό) χωρίζονται σε κομμάτια των 8x8 pixels.
2. Για κάθε κομμάτι υπολογίζεται ο μέσος όρος των τιμών του Cb και του Cr χωριστά δίνοντας μεγαλύτερο βάρος (3/2) στο δεύτερο.
3. Υπολογίζονται οι απόλυτες διαφορές για όλα τα κομμάτια του αρχικού από το παραμορφωμένο.

4. Υπολογίζεται για κάθε καρέ η τυπική απόκλιση [11].
5. Οι τυπικές αποκλίσεις ταξινομούνται από το μικρότερο στο μεγαλύτερο και εξάγεται η τιμή που βρίσκεται στο 10% της ταξινόμησης.
6. Η τιμή πολλαπλασιάζεται με 0.0192.

Εικόνα 21: Παράδειγμα χρωματικών απωλειών.

4.2.2 Απώλειες καρέ

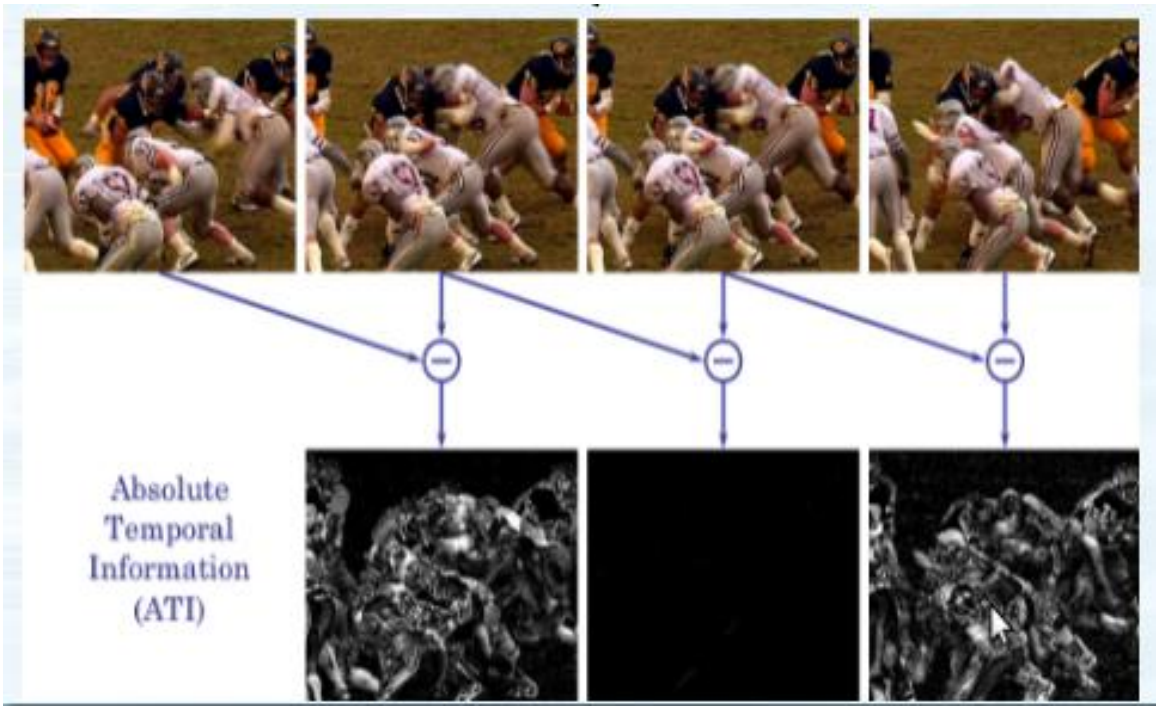
Απώλειες καρέ παρουσιάζονται συνήθως σε περιπτώσεις που φαίνεται να «παγώνει» η εικόνα. Με τον ίδιο τρόπο μπορεί να ανιχνευθεί και το αντίστροφο φαινόμενο δηλ. η μη επιθυμητή κίνηση που μπορεί να έχει προστεθεί στο παραμορφωμένο βίντεο. Τέτοια φαινόμενα συνήθως δημιουργούνται από σφάλματα στην μετάδοση. Όταν χάνονται πακέτα ο αποκωδικοποιητής δεν μπορεί να ανανεώσει την εικόνα ή κάποια μέρη της εικόνας και έτσι παγώνει. Όμως (σπάνια) μπορούν να δημιουργηθούν και κατά το στάδιο της συμπίεσης, με αποτέλεσμα το βίντεο που θα μεταδοθεί να είναι εξ' αρχής εσφαλμένο [12].

Οι απώλειες καρέ στο άθροισμα του VQM score εκφράζονται από τον όρο:

+0.0431*Y_contrast_ati_4x4_6F_std_ratio_gain_mean_10%

Για τον υπολογισμό του όρου, εκτελούνται τα εξής:

1. Τα βίντεο (αρχικό και τελικό) χωρίζονται σε κομμάτια των 4x4 pixels x 6 frames.
2. Για κάθε κομμάτι υπολογίζεται η τυπική απόκλιση για το contrast (αντίθεση) και το ati (Απόλυτη Χρονική Πληροφορία – Absolute Temporal Information) χωριστά. Το contrast είναι απλά το σήμα της φωτεινότητας ενώ το ati είναι η απόλυτη διαφορά του pixel της φωτεινότητας από το pixel που βρίσκεται στο προηγούμενο καρέ.
3. Υπολογίζεται το contrast ati που είναι το γινόμενό τους.
4. Υπολογίζεται η σχετική διαφορά του αρχικού βίντεο σε σχέση με το παραμορφωμένο. Όσες τιμές είναι κάτω από μηδέν μηδενίζονται.
5. Εξάγεται η μέση τιμή για κάθε καρέ.
6. Οι τυπικές αποκλίσεις ταξινομούνται από το μικρότερο στο μεγαλύτερο και εξάγεται η τιμή που βρίσκεται στο 10% της ταξινόμησης.
7. Η τιμή πολλαπλασιάζεται με 0.0431.



Εικόνα 22: Το πάγωμα της εικόνας εκφράζεται από την απουσία χωρικής πληροφορίας ATI.

4.2.3 Μέτρηση Blurring

Το θόλωμα (blurring) εκφράζει το πόσο θολό είναι το παραμορφωμένο βίντεο σε σχέση με το αρχικό. Στην ψηφιακή τηλεόραση, το θόλωμα είναι αποτέλεσμα υπερβολικής συμπίεσης, δηλαδή χρησιμοποίηση μεγάλης κλίμακας κβάντισης κατά τον μετασχηματισμό DCT. Συνήθως συνδυάζεται με την εμφάνιση θορύβου στο παραμορφωμένο βίντεο. Το θόλωμα είναι δύσκολο να εκτιμηθεί υποκειμενικά επειδή εξαρτάται από τον φωτισμό που υπάρχει στον εξωτερικό χώρο.

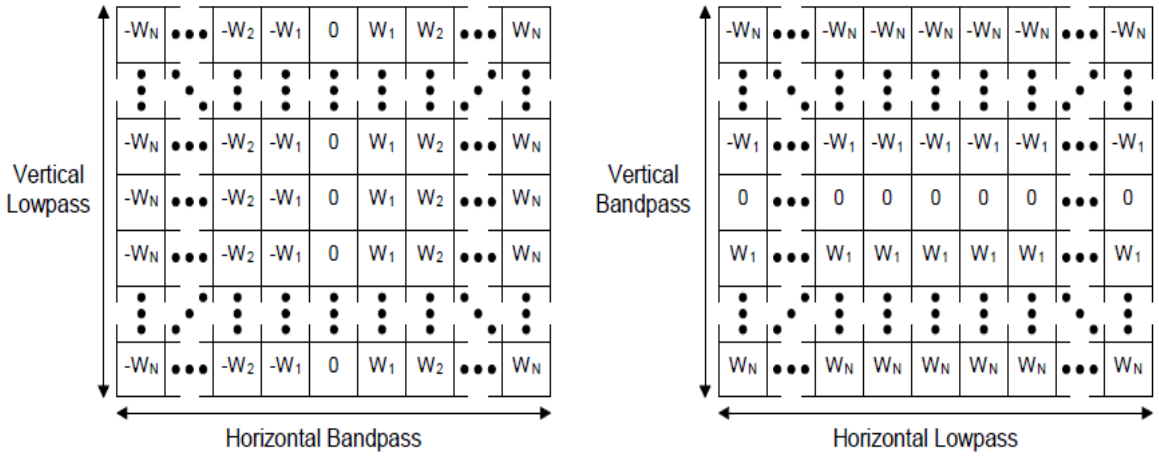
Το θόλωμα (blurring) στο άθροισμα του VQM score εκφράζεται από τον όρο:

-0.2097*Y_si13_8x8_6F_std_12_ratio_loss_below5%

Εκτελούνται τα εξής:

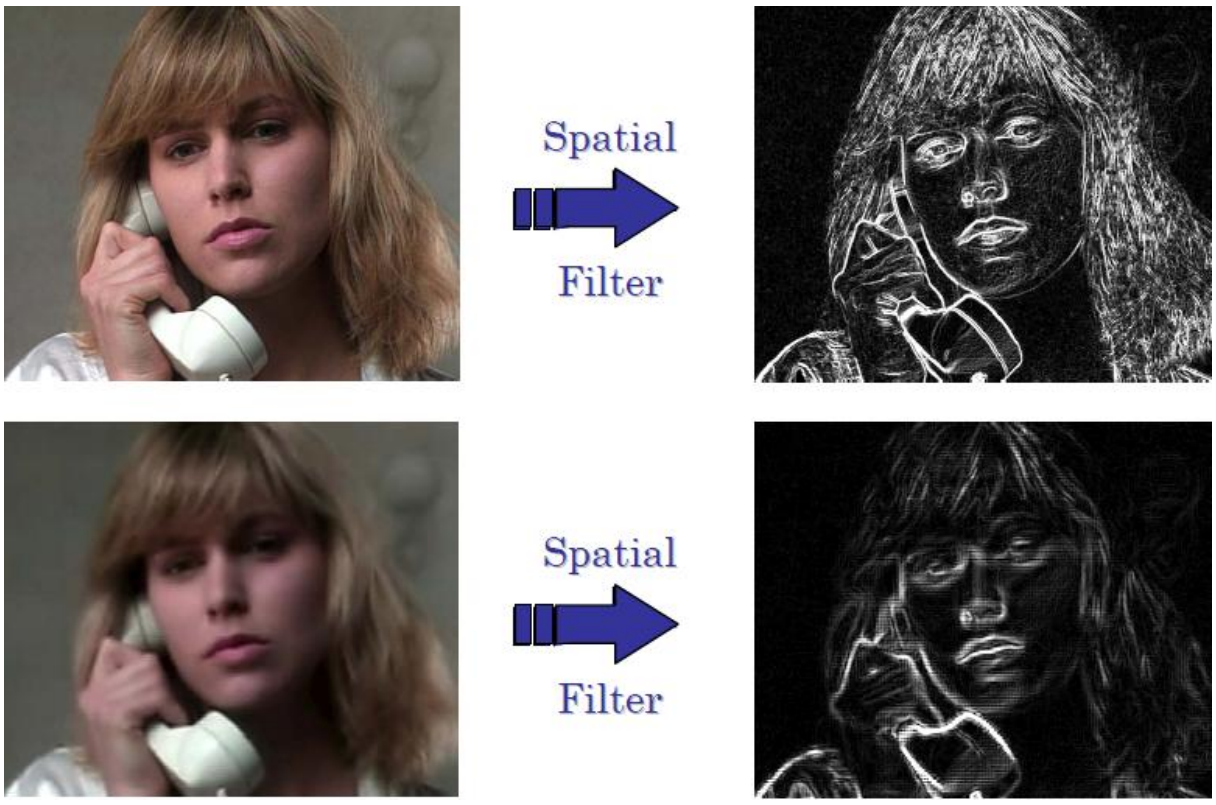
1. Τα βίντεο (αρχικό και τελικό) χωρίζονται σε κομμάτια των 8x8 pixels x 6 frames.
2. Εφαρμόζονται οριζόντια H και κάθετα V φίλτρα, δηλαδή το άθροισμα των 13x13 γειτονικών pixels, δίνοντας διαφορετικό βάρος στη κάθε στήλη και κάθε γραμμή αντίστοιχα. Αυτό γίνεται για κάθε pixel. Τα βάρη είναι [-0.0052625, -0.0173446, -0.0427401, -0.0768961, -0.0957739, -0.0696751, 0, 0.0696751, 0.0957739, 0.0768961, 0.0427401, 0.0173446, 0.0052625] αρχίζοντας από αριστερά προς τα δεξιά και από πάνω προς τα κάτω αντίστοιχα [13].
3. Τοποθετούνται τα αποτελέσματα των φίλτρων σε ένα πολικό σύστημα συντεταγμένων. Το οριζόντιο φίλτρο στον x άξονα και το κάθετο στον y άξονα.
4. Υπολογίζεται η ακτινική απόσταση R από την αρχή των αξόνων.
5. Το SI είναι η τυπική απόκλιση των R για κάθε κομμάτι.
6. Όσες τιμές είναι μικρότερες από 12 γίνονται 12.
7. Υπολογίζεται η σχετική διαφορά του αρχικού σε σχέση με το παραμορφωμένο. Όσες τιμές είναι πάνω από μηδέν μηδενίζονται.

8. Υπολογίζεται ο μέσος όρος για το 5% των μικρότερων τιμών.
9. Το αποτέλεσμα πολλαπλασιάζεται με -0.2097.



Εικόνα 23: Εφαρμογή οριζόντιων και κάθετων φίλτρων, όπου w το αντίστοιχο βάρος.

Το αριστερό μεγαλώνει τις διαφορές των pixel οριζόντια ενώ εξομαλύνει κάθετα και το δεξιό μεγαλώνει τις διαφορές των pixel κάθετα ενώ εξομαλύνει οριζόντια.



Εικόνα 24: Απώλεια της οξύτητας της εικόνας (θόλωμα) όπως φαίνεται μετά την εφαρμογή του φίλτρου.

4.2.4 Μέτρηση Enhancement

Το enhancement αφορά την αύξηση της οξύτητας (sharpness) της εικόνας που γίνεται στο δέκτη με την εφαρμογή φίλτρων sharpness (όπως π.χ. στην πλειονότητα των σύγχρονων

τηλεοράσεων). Επίσης κάποιοι κωδικοποιητές εικόνας για να μειώσουν τη θολούρα που ενδεχόμενα υπάρχει στο αρχικό βίντεο χρησιμοποιούν φίλτρα enhancement και κάνουν την εικόνα πιο οξεία. Θεωρείται θετικό metric, γι' αυτό βελτιώνει το VQM.

Ο παράγοντας enhancement στο άθροισμα του VQM score εκφράζεται από τον όρο:

+2.3416*Y_si13_8x8_6F_std_8_log_gain_mean

Για τον υπολογισμό του εκτελούνται τα εξής:

1. Τα βίντεο (αρχικό και τελικό) χωρίζονται σε κομμάτια των 8x8 pixels x 6 frames.
2. Εφαρμόζονται οριζόντια H και κάθετα V φίλτρα, δηλαδή το άθροισμα των 13x13 γειτονικών pixels, δίνοντας διαφορετικό βάρος στη κάθε στήλη και κάθε γραμμή αντίστοιχα. Αυτό γίνεται για κάθε pixel.
3. Τοποθετούνται τα αποτελέσματα των φίλτρων σε ένα πολικό σύστημα συντεταγμένων. Το οριζόντιο φίλτρο στον x άξονα και το κάθετο στον y άξονα.
4. Υπολογίζεται η ακτινική απόσταση R από την αρχή των αξόνων.
5. Το SI είναι η τυπική απόκλιση των R για κάθε κομμάτι.
6. Όσες τιμές είναι μικρότερες από 8 γίνονται 8.
7. Υπολογίζεται ο λόγος σε λογαριθμική κλίμακα του αρχικού σε σχέση με το παραμορφωμένο. Όσες τιμές είναι κάτω από μηδέν μηδενίζονται.
8. Εξάγεται ο μέσος όρος των παραπάνω αποτελεσμάτων.
9. Το αποτέλεσμα πολλαπλασιάζεται με 2.3416.

Εικόνα 25: Ο αλγόριθμος συμπίεσης στο παράδειγμα έχει καταφέρει να αυξήσει τη λεπτομέρεια στο τελικό βίντεο μειώνοντας τη θολούρα.

4.2.5 Μέτρηση Blockiness

Ο παράγοντας αυτός εκτιμάει το λεγόμενο «blockiness effect» στο παραμορφωμένο βίντεο. Είναι και αυτό αποτέλεσμα υπερβολικής συμπίεσης. Όπως προαναφέρθηκε, με τη χρήση υψηλού συντελεστή κβάντισης κατά την κωδικοποίηση, τα block παραμορφώνονται, χάνουν τη λεπτομέρεια τους αλλά και τη συνέχειά τους και γίνονται ορατά τα όρια των blocks. Μπορεί να συμβεί όμως και από σφάλματα κατά τη μετάδοση όταν δεν έχουν ανανεωθεί μεμονωμένα κομμάτια από την εικόνα.

Η επιρροή του blockiness effect στο άθροισμα του VQM score εκφράζεται από τον όρο:

+0.2483*Y_hv13_angle0.225_rmin20_8x8_6F_mean_3_log_gain_above95%_mean

Εκτελούνται τα εξής:

1. Τα βίντεο (αρχικό και τελικό) χωρίζονται σε κομμάτια των 8x8 pixels x 6 frames.
2. Εφαρμόζονται οριζόντια H και κάθετα V φίλτρα, δηλαδή το άθροισμα των 13x13 γειτονικών pixels, δίνοντας διαφορετικό βάρος στη κάθε στήλη και κάθε γραμμή αντίστοιχα. Αυτό γίνεται για κάθε pixel.
3. Τοποθετούνται τα αποτελέσματα των φίλτρων σε ένα πολικό σύστημα συντεταγμένων. Το οριζόντιο φίλτρο στον x άξονα και το κάθετο στον y άξονα.
4. Υπολογίζεται η ακτινική απόσταση R από την αρχή των αξόνων και η δεξιόστροφη γωνία θ(ή αλλιώς πολικός άξονας) [14] με τους τύπους:

$$R(i,j,t)=\sqrt{H(i,j,t)^2+V(i,j,t)^2}$$

$$\theta(i,j,t)=\tan^{-1}\left[\frac{V(i,j,t)}{H(i,j,t)}\right]$$

5. Υπολογίζονται τα HV και $\overline{HV}$. Στο HV, pixels που είναι διαγώνιες άκρες μηδενίζονται. Στο $\overline{HV}$, pixels που είναι οριζόντιες ή κάθετες άκρες μηδενίζονται. Αυτό ρυθμίζεται ανάλογα με την τιμή του θ. Μετά υπολογίζεται το hv13 που είναι ο λόγος της μέσης τιμής για κάθε κομμάτι των HV και $\overline{HV}$.

$$HV(i,j,t)=\begin{cases} R(i,j,t) & if\ R(i,j,t)\geq r_{\min}\ and\ m\frac{\pi}{2}-\Delta\theta<\theta(i,j,t)<m\frac{\pi}{2}+\Delta\theta \quad (m=0,1,2,3) \\ 0 & otherwise \end{cases},$$

and

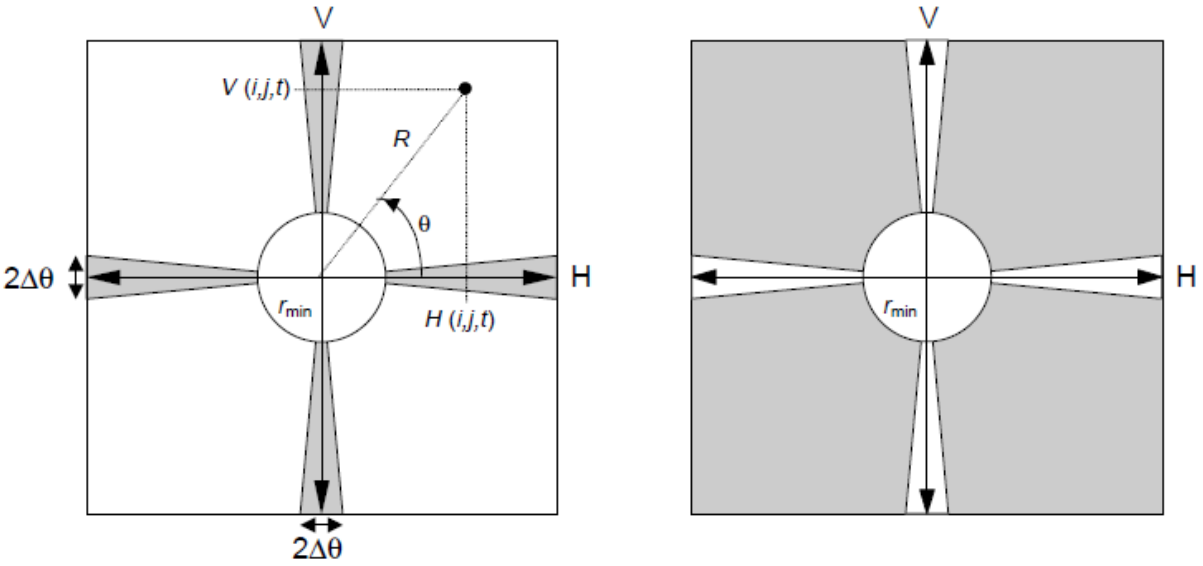
$$\overline{HV}(i,j,t)=\begin{cases} R(i,j,t) & if\ R(i,j,t)\geq r_{\min}\ and\ m\frac{\pi}{2}+\Delta\theta\leq\theta(i,j,t)\leq(m+1)\frac{\pi}{2}-\Delta\theta \quad (m=0,1,2,3) \\ 0 & otherwise \end{cases}$$

Όπου:

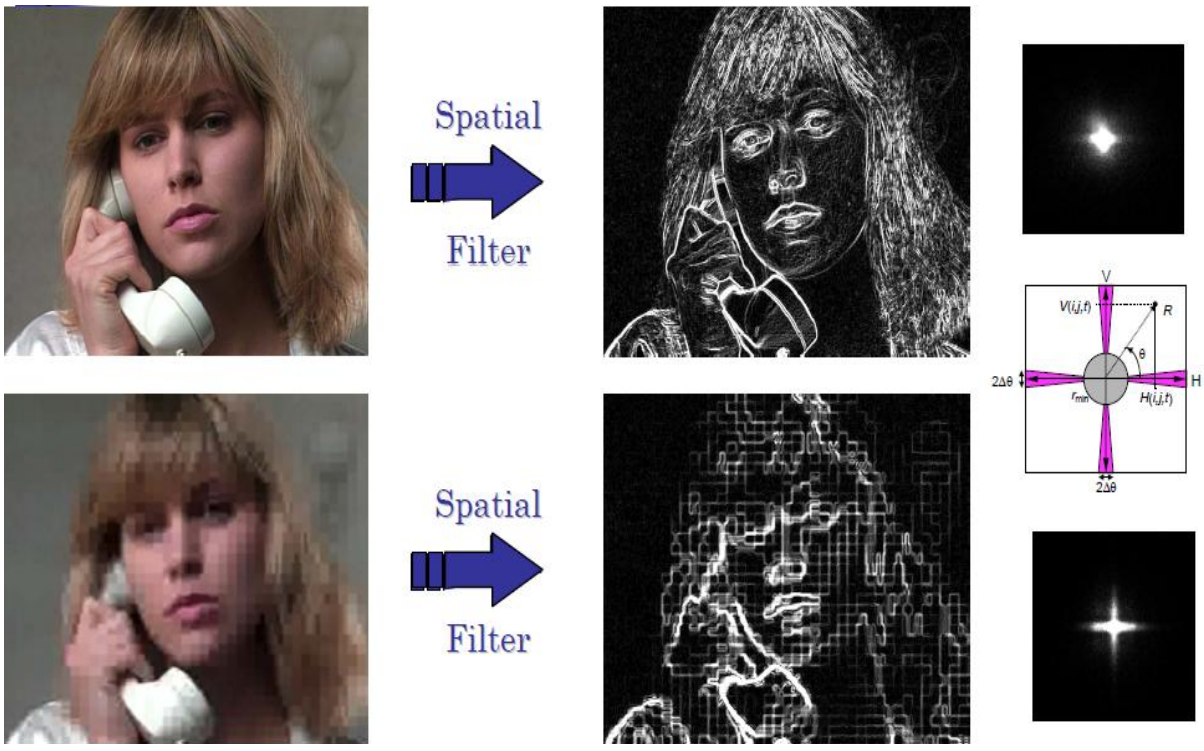
- $\Delta\theta$=0.225 radians.
- r_{min}=20.

6. Όσες τιμές είναι μικρότερες από 3 γίνονται 3.

7. Υπολογίζεται ο λόγος σε λογαριθμική κλίμακα του αρχικού σε σχέση με το παραμορφωμένο. Όσες τιμές είναι κάτω από μηδέν μηδενίζονται.
8. Υπολογίζεται ο μέσος όρος για το 5% των μεγαλύτερων τιμών για κάθε ομάδα καρέ.
9. Ο μέσος όρος πολλαπλασιάζεται με 0.2483.



Εικόνα 26: Διαχωρισμός του HV (αριστερή εικόνα) και $\overline{HV}$ (δεξιά εικόνα).



Εικόνα 27: Εμφάνιση του blockiness effect όπως φαίνεται μετά την εφαρμογή του φίλτρου.

4.2.6 Μέτρηση Smearing

Το φαινόμενο χρωματικής διασποράς (smearing) στο παραμορφωμένο βίντεο είναι κάτι αντίστοιχο με το θόλωμα. Συνήθως δημιουργείται σε χωρικά πολύπλοκες σκηνές.

Κάποια αντικείμενα επικαλύπτονται από κάποια γειτονικά τους. Χάνεται δηλαδή η ευκρίνεια και η λεπτομέρεια στην εικόνα. Πρόκειται για χαρακτηριστική παραμόρφωση που δημιουργείται από υπερβολική συμπίεση. Το smearing μπορεί να εμφανιστεί ακόμα, όταν χρησιμοποιείται η μέθοδος πεπλεγμένης σάρωσης και υπάρξουν σφάλματα μετάδοσης σε ένα από τα δύο διαδοχικά καρέ.

Το φαινόμενο smearing στο άθροισμα του VQM score εκφράζεται από τον όρο:

+0.5969*Y_hv13_angle0.225_rmin20_8x8_6F_mean_3_ratio_loss_below5%_mean_square

Εκτελούνται τα εξής:

1. Τα βίντεο (αρχικό και τελικό) χωρίζονται σε κομμάτια των 8x8 pixels x 6 frames.
2. Εφαρμόζονται οριζόντια H και κάθετα V φίλτρα, δηλαδή το άθροισμα των 13x13 γειτονικών pixels, δίνοντας διαφορετικό βάρος στη κάθε στήλη και κάθε γραμμή αντίστοιχα. Αυτό γίνεται για κάθε pixel.
3. Τοποθετούνται τα αποτελέσματα των φίλτρων σε ένα πολικό σύστημα συντεταγμένων. Το οριζόντιο φίλτρο στον x άξονα και το κάθετο στον y άξονα.
4. Υπολογίζεται η ακτινική απόσταση R από την αρχή των αξόνων και η δεξιόστροφη γωνία θ.
5. Υπολογίζονται τα HV και $\overline{HV}$. Στο HV, pixels που είναι διαγώνιες άκρες μηδενίζονται. Στο $\overline{HV}$, pixels που είναι οριζόντιες ή κάθετες άκρες μηδενίζονται. Αυτό ρυθμίζεται ανάλογα με την τιμή του θ. Μετά υπολογίζεται το hv13 που είναι ο λόγος της μέσης τιμής για κάθε κομμάτι των HV και $\overline{HV}$.
6. Όσες τιμές είναι μικρότερες από 3 γίνονται 3.
7. Υπολογίζεται η σχετική διαφορά του αρχικού σε σχέση με το παραμορφωμένο. Όσες τιμές είναι πάνω από μηδέν μηδενίζονται.
8. Υπολογίζεται ο μέσος όρος για το 5% των μικρότερων τιμών για κάθε ομάδα καρέ.
9. Ο μέσος όρος υψώνεται στο τετράγωνο και πολλαπλασιάζεται με 0.5969.

Εικόνα 28: Παραμόρφωση smearing.

4.3 Σύγκριση VQM με PSNR

Έχοντας περιγράψει αναλυτικά τους αλγορίθμους PSNR και VQM για εκτίμηση υποκειμενικής ποιότητας εικόνας, όσον αφορά τη μεταξύ τους σύγκριση μπορούν να αναφερθούν τα εξής:

1. Το PSNR είναι απλό στην χρήση και στην υλοποίηση. Το VQM είναι πολύπλοκο επειδή υπολογίζει έξι ανεξάρτητους μεταξύ τους παράγοντες.
2. Το PSNR είναι πολύ γρήγορο στην εκτέλεση σε σχέση με το VQM. Το VQM χρειάζεται μεγάλη επεξεργαστική ισχύ και μνήμη για να λειτουργήσει σωστά και να παρουσιάσει τα αποτελέσματα σε εύλογο χρονικό διάστημα.
3. Το VQM αντιλαμβάνεται την παραμόρφωση όπως η ανθρώπινη όραση. Το PSNR μετράει μόνο την απώλεια ή προσθήκη ενέργειας στα bytes της φωτεινότητας.
4. Το VQM πραγματοποιεί μετρήσεις στην φωτεινότητα αλλά και στα χρώματα κόκκινο και μπλε του YUV συστήματος. Το PSNR υπολογίζεται μόνο με βάση τη φωτεινότητα.
5. Το VQM εντοπίζει το είδος του προβλήματος. Πχ. παραμορφώσεις, απώλειες, blockiness, πάγωμα εικόνας.

5 Λειτουργία του προγράμματος VQM & PSNR analyser

Εκτός από τη μελέτη των αλγορίθμων VQM και PSNR, σκοπός της διπλωματικής ήταν η υλοποίησή τους και η δοκιμή με συγκεκριμένα video. Η υλοποίηση τόσο του VQM και του PSNR έγινε σε γλώσσα C++ (για μεγαλύτερη αποδοτικότητα) και σε περιβάλλον Linux. Ο κώδικας που αναπτύχθηκε περιλαμβάνεται για αναφορά στο Παράρτημα.

Η ενότητα αυτή όπως και οι δύο επόμενες περιγράφουν τα προαπαιτούμενα και τη λειτουργία του προγράμματος που αναπτύχθηκε.

5.1 Προαπαιτούμενα

1. Το πρόγραμμα είναι Full Reference, δηλαδή χρειάζεται να έχουμε το βίντεο που παράγεται στον τηλεοπτικό σταθμό και το βίντεο που βλέπει ο τηλεθεατής.
2. Τα βίντεο πρέπει να έχουν την ίδια ανάλυση, ίδιο ρυθμό καρέ.
3. Το ύψος και το πλάτος της ανάλυσης να είναι πολλαπλάσια του 8 (ισχύει σε όλα τα κωδικοποιημένα κατά MPEG video)
4. Τα βίντεο να είναι ασυμπίεστα τύπου UYVY χωρίς container (παρακάτω περιγράφεται το format).
5. Το πρόγραμμα εκτελείται σε λειτουργικό σύστημα LINUX UBUNTU.
6. Απαιτείται επαρκής μνήμη RAM, πάνω από 1 GByte το ελάχιστο, για βίντεο pal των 30 sec.
7. Απαιτείται αρκετή επεξεργαστική ισχύ, ενδεικτικά επεξεργαστή χρονισμένο άνω των 2 GHz για να εμφανίζει τα αποτελέσματα σε λογικά χρονικά πλαίσια.
8. Το μέγεθος του αρχικού βίντεο να είναι μεγαλύτερο ή ίσο με το παραμορφωμένο.

5.2 Δοκιμαστική εκτέλεση

1. Στο Linux ανοίγουμε παράθυρο γραμμής εντολών (Terminal).

2. Μεταφερόμαστε στον φάκελο που έχουμε αποθηκεύσει το πρόγραμμα. Πχ. Αν το είχαμε αποθηκεύσει στο desktop θα δίναμε την εντολή:

   cd '/home/sarafo/Desktop/VQM & PSNR (UYVY videos) (v1)'

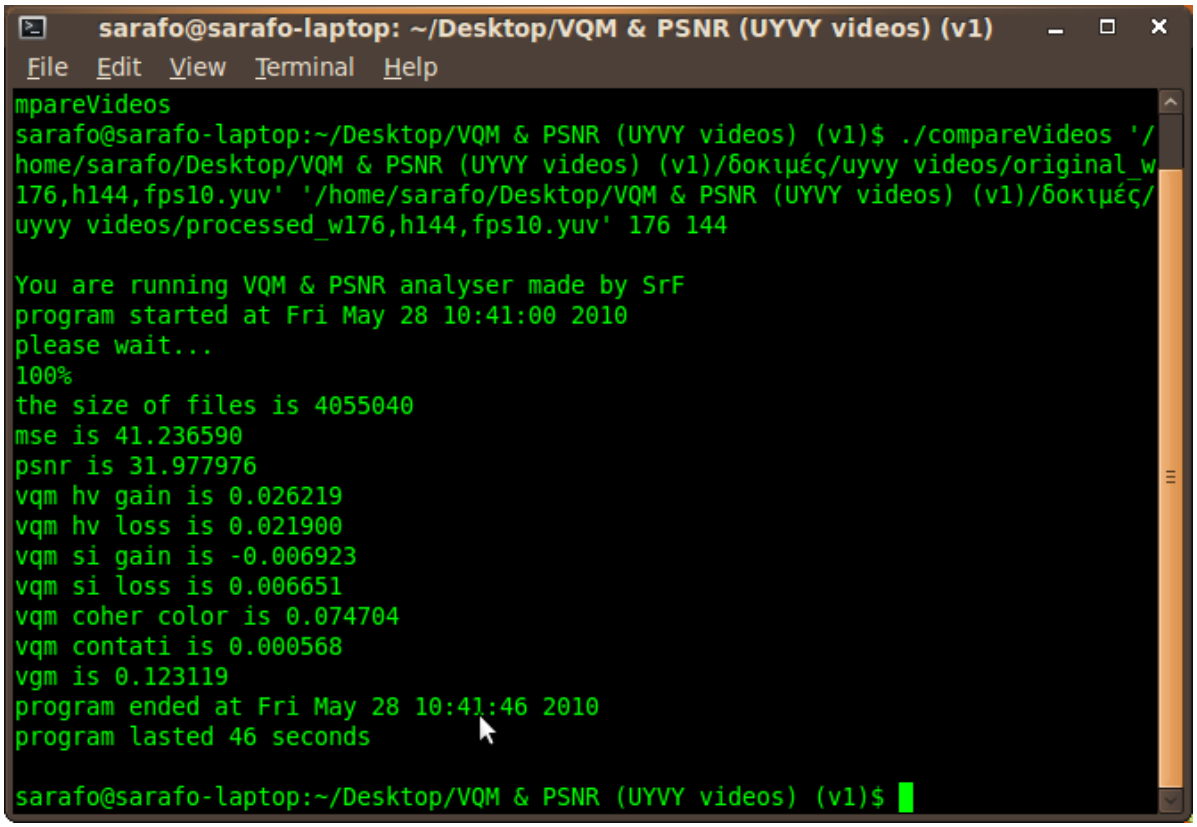
3. Τρέχουμε την εντολή ./compareVideos.

Το πρόγραμμα ενημερώνει για το format που πρέπει να δώσουμε τις παραμέτρους για να λειτουργήσει σωστά και ζητάει τη διεύθυνση του αρχικού βίντεο, τη διεύθυνση του παραμορφωμένου βίντεο, το πλάτος και το ύψος των δύο βίντεο. Αν δεν δοθεί σωστά η ανάλυση ή εμφανιστεί κάποιο πρόβλημα κατά το άνοιγμα των αρχείων θα εμφανιστεί το αντίστοιχο σφάλμα. Εκτός των άλλων το πρόγραμμα εμφανίζει το μέγεθος των αρχείων, την ώρα που ξεκίνησε και την ώρα που τελείωσε η ανάλυση.

4. Δίνουμε τα παραπάνω στοιχεία όπως στο παράδειγμα που ακολουθεί:

./compareVideos '/home/sarafo/Desktop/VQM & PSNR (UYVY videos) (v1)/original_w176,h144,fps10.yuv' '/home/sarafo/Desktop/VQM & PSNR (UYVY videos) (v0.8)/processed_w176,h144,fps10.yuv' 176 144

5. Μετά την επεξεργασία των βίντεο, το πρόγραμμα παρουσιάζει τα αποτελέσματα όπως φαίνεται στο screenshot που ακολουθεί. Φαίνεται τόσο το PSNR σαν τιμή, ο καθένας από τους παράγοντες του VQM καθώς και το συνολικό VQM score.



Εικόνα 29: Δοκιμαστική εκτέλεση VQM.

6 Δοκιμές

Οι δοκιμές έγιναν με την εφαρμογή των αλγορίθμων VQM και PSNR πάνω σε ένα σύνολο βίντεο που είχαν υποστεί παραμορφώσεις. Τα βίντεο αυτά ήταν διάρκειας λίγων δευτερολέπτων και κωδικοποιημένα κατά MPEG-4.

Επειδή η πτυχιακή εστιάζεται στη μετάδοση ψηφιακού βίντεο πάνω από δίκτυα ψηφιακής τηλεόρασης ή/και δεδομένων, η κύρια παραμόρφωση (impairment) που υπεισέρχεται είναι η απώλεια κάποιων δεδομένων λόγω προβλημάτων/διαλείψεων στη μετάδοση.

Προκειμένου να εξομοιωθεί το φαινόμενο αυτό, αναπτύχθηκε ειδικό λογισμικό (βλ. επόμενη ενότητα «Πρόγραμμα AddSpaces» που προκαλεί εκούσια απώλεια δεδομένων σε αρχεία βίντεο. Έτσι, κατά την αναπαραγωγή του επεξεργασμένου αρχείου, προσεγγίζεται το φαινόμενο της απώλειας δεδομένων σε ένα σύστημα ψηφιακής τηλεόρασης.

Να σημειωθεί ότι τόσο το αρχικό όσο και το παραμορφωμένο βίντεο, προκειμένου να υποστούν ανάλυση από το πρόγραμμα εκτίμησης VQM/PSNR που αναπτύχθηκε πρέπει να μετατραπούν πρώτα σε ασυμπίεστη μορφή UYVY. Το format αυτό, καθώς και το πρόγραμμα που χρησιμοποιήθηκε για τη μετατροπή (mencoder) περιγράφονται στο παράρτημα.

Η αναπαραγωγή των βίντεο έγινε με το πρόγραμμα mplayer, που επίσης περιγράφεται συνοπτικά στο παράρτημα για λόγους πληρότητας.

6.1 Πρόγραμμα Add Spaces

6.1.1 Περιγραφή

Πρόκειται για πρόγραμμα που εξομοιώνει την παραμόρφωση που εισάγεται σε ένα βίντεο, μετά την μετάδοση σε χαμηλό σήματοθορυβικό λόγο SNR. Το πρόγραμμα αυτό αναπτύχθηκε στα πλαίσια της πτυχιακής για τις ανάγκες των δοκιμών. Με τρόπο ντετερμινιστικό, αφαιρεί από το κωδικοποιημένο αρχείο βίντεο ένα block των N bytes. Η αφαίρεση γίνεται περιοδικά κάθε M bytes. Οι παράμετροι N και M καθορίζονται από τον χρήστη.

6.1.2 Χρήση

Το πρόγραμμα καλείται από το περιβάλλον τερματικού του Linux, εκτελώντας ./addSpaces. Όλοι οι παράμετροι περνιούνται στο command line. Στις παραμέτρους αυτές (που πρέπει να οριστούν) περιλαμβάνονται:

- Η θέση και το όνομα του αρχικού βίντεο.
- Η θέση και το όνομα του παραμορφωμένου βίντεο που θα φτιάξει.
- Κάθε πόσα bytes να εισάγει παραμόρφωση (M) και
- το μέγεθος σε bytes της παραμόρφωσης (N).

πχ.

./addSpaces '/home/sarafo/Desktop/mp4 videos/video_1.mp4' video_1_disorted.mp4 10000 100

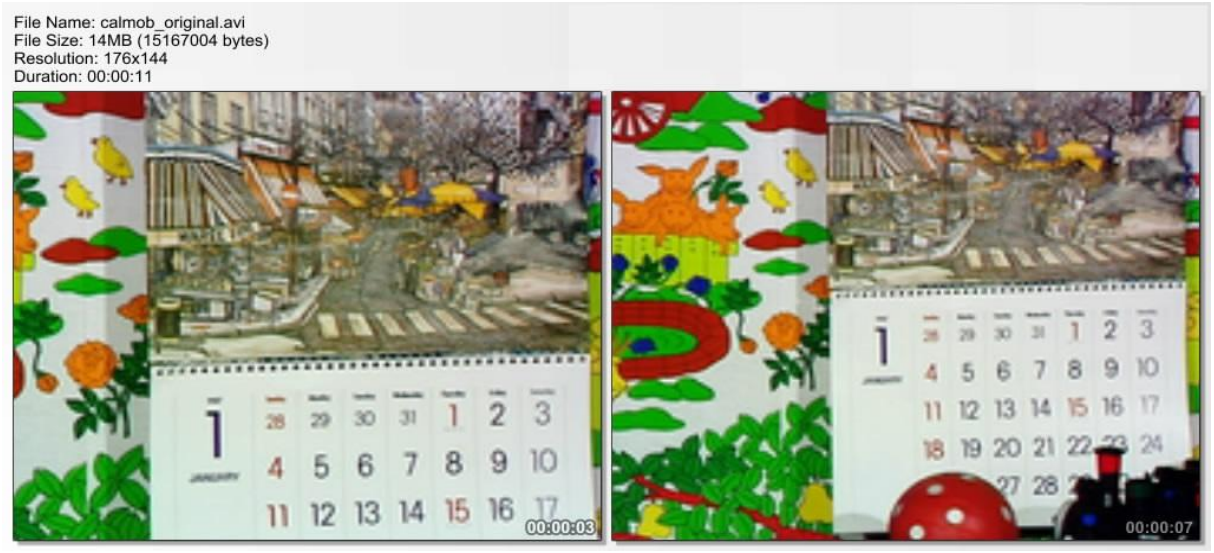
Στη συνέχεια περιγράφονται οι δοκιμές που έγιναν και που περιελάμβαναν βίντεο ποικίλου περιεχομένου, με επίσης ποικίλο ποσοστό απωλειών ώστε να είναι αντιπροσωπευτικές.

Για κάθε δοκιμή αναφέρονται:

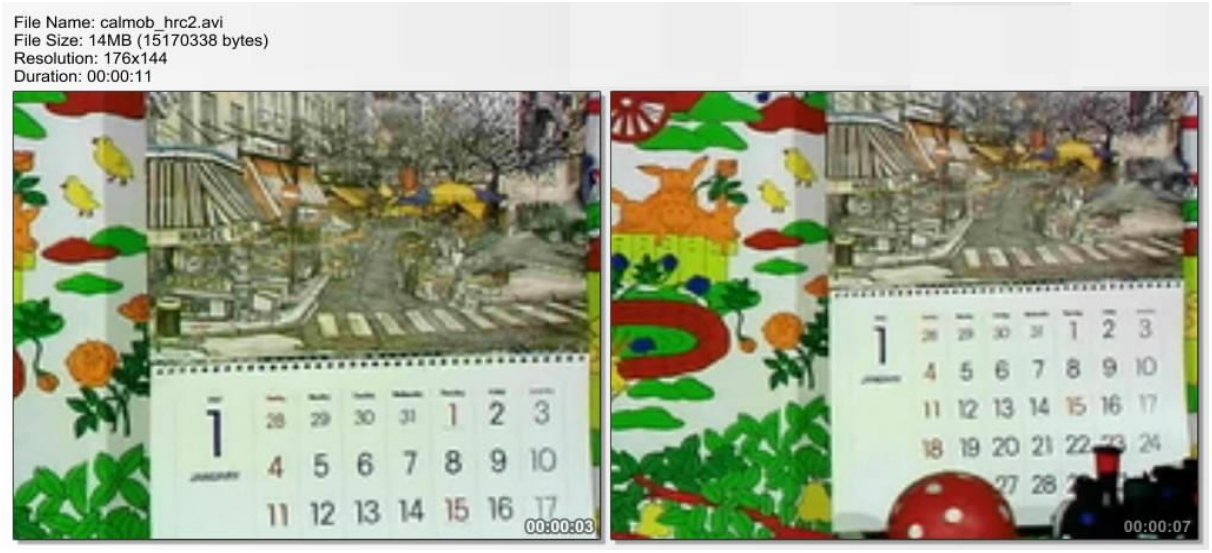
- Το περιεχόμενο του βίντεο (παρουσιάζονται τέσσερα screenshots).
- Η ανάλυση, το frame rate και η διάρκειά του.
- Οι παράμετροι των απωλειών (M και N) που εισήχθησαν με το πρόγραμμα AddSpaces.
- Τα αποτελέσματα του PSNR metric, όπως υπολογίστηκαν από το λογισμικό που αναπτύχθηκε.
- Τα αποτελέσματα του VQM metric, όπως υπολογίστηκαν από το λογισμικό που αναπτύχθηκε. Να σημειωθεί ξανά ότι όσο μεγαλύτερο είναι το VQM metric, τόσο εντονότερες είναι οι παραμορφώσεις σε μια υποκειμενική εκτίμηση.

6.2 Πρώτη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 176x144, Fps: 25, Διάρκεια: 12 seconds., Απώλειες -> Κυρίως λόγο συμπίεσης.



Εικόνα 30: Βίντεο calmob αρχικό screenshots.



Εικόνα 31: Βίντεο calmob παραμορφωμένο screenshots.

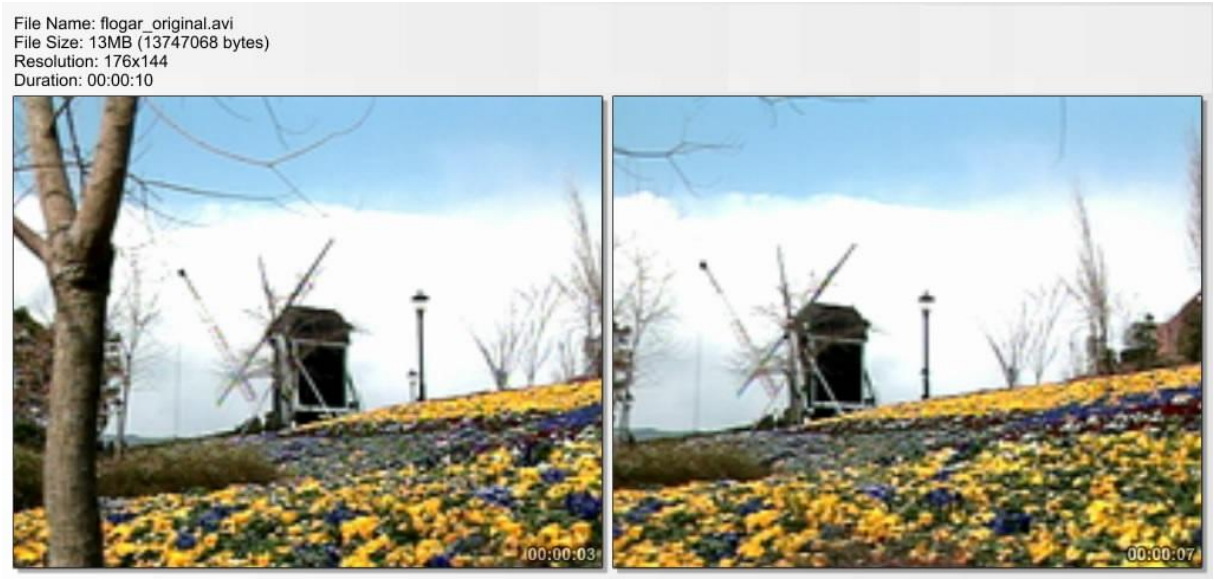
Αριθμός δοκιμής	Πρώτη
Διάρκεια Εκτέλεσης Ανάλυσης	176 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	calmob
Μέγεθος αρχείου	14 Mega Bytes
Πλάτος	176 pixels
Ύψος	144 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	12 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	-
Μέγεθος απώλειας σε bytes	-
Αποτελέσματα Δοκιμής:	
MSE	159,402444
PSNR	26,105854
VQM HV GAIN (μέτρηση Blockiness)	0,062134
VQM HV LOSS (μέτρηση Smearing)	0,075268
VQM SI GAIN (μέτρηση Enchancement)	-0,014250
VQM SI LOSS (μέτρηση Blurring)	0,031759
VQM COHER COLOR (Χρωματικές απώλειες)	0,045978
VQM CONTATI (Απώλειες καρέ)	0,001714
VQM GENERAL	0,202603

Πίνακας 1: Πρώτη δοκιμή.

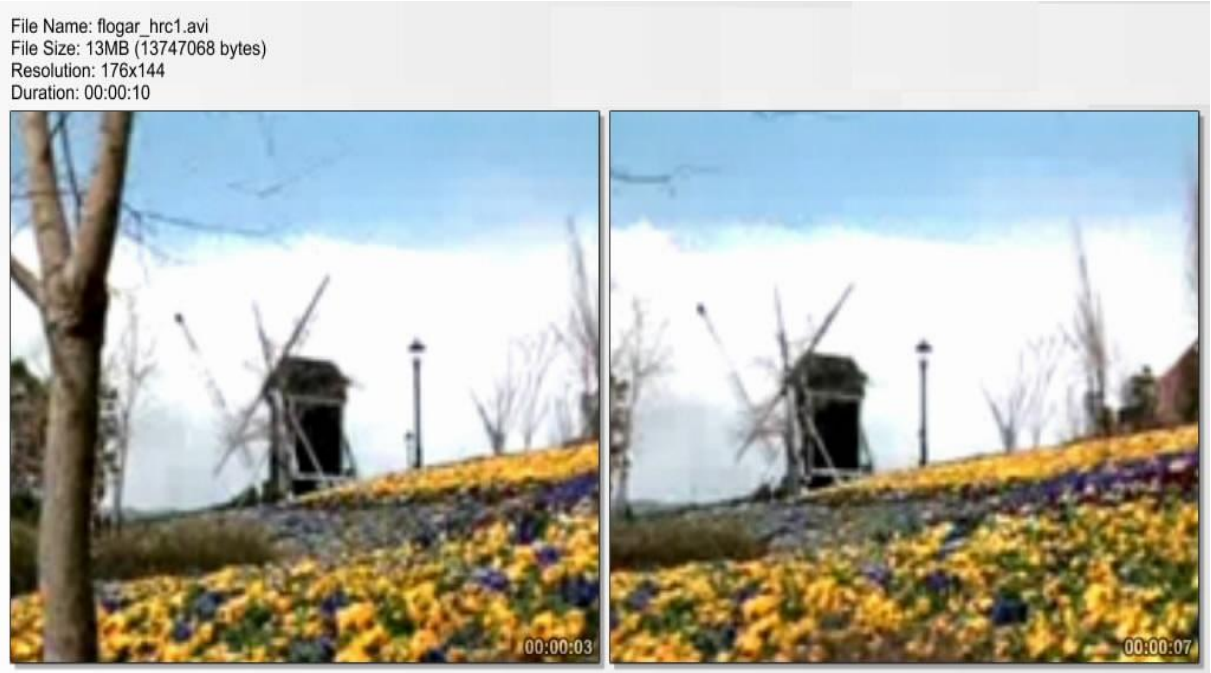
Σ' αυτή τη δοκιμή, οι απώλειες ήταν αποτέλεσμα κυρίως λανθασμένης ρύθμισης του κωδικοποιητή εικόνας. Στο παραμορφωμένο βίντεο έχει μειωθεί η ζωντάνια των χρωμάτων, έχει ελαττωθεί η φωτεινότητα και σε κάποια αντικείμενα έχει χαθεί η λεπτομέρεια. Όμως η παραμόρφωση είναι σχετικά μικρή και το VQM έχει δώσει υψηλή βαθμολογία (μικρό VQM).

6.3 Δεύτερη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 176x144, Fps: 25, Διάρκεια: 10.8 seconds. Ιδιότητες Απωλειών-> Κυρίως λόγο συμπίεσης.



Εικόνα 32: Βίντεο flogar αρχικό screenshots.



Εικόνα 33: Βίντεο flogar παραμορφωμένο screenshots.

Αριθμός δοκιμής	Δεύτερη
Διάρκεια Εκτέλεσης Ανάλυσης	161 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	flogar
Μέγεθος αρχείου	13 Mega Bytes
Πλάτος	176 pixels
Ύψος	144 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	10,8 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	-
Μέγεθος απώλειας σε bytes	-
Αποτελέσματα Δοκιμής:	
MSE	439,622536
PSNR	21,700004
VQM HV GAIN (μέτρηση Blockiness)	0,133822
VQM HV LOSS (μέτρηση Smearing)	0,208623
VQM SI GAIN (μέτρηση Enchancement)	-0,023199
VQM SI LOSS (μέτρηση Blurring)	0,047060
VQM COHER COLOR (Χρωματικές απώλειες)	0,034246
VQM CONTATI (Απώλειες καρέ)	0,000902
VQM GENERAL	0,401454

Πίνακας 2: Δεύτερη δοκιμή.

Στο βίντεο αυτό, αν και οι απώλειες είναι παρόμοιες με την πρώτη δοκιμή, η παραμόρφωση είναι μεγαλύτερη. Κάποιες λεπτομέρειες έχουν χαθεί εξ' ολοκλήρου (όπως τα κλαδιά των δέντρων). Αυτό συμβαίνει εξαιτίας του πανοραμικού τοπίου. Οι πανοραμικές σκηνές χρειάζονται λιγότερα bytes για την αναπαράστασή τους, με αποτέλεσμα οι απώλειες να επιδρούν περισσότερο.

6.4 Τρίτη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 176x144, Fps: 10, Διάρκεια: 8 seconds. Ιδιότητες Απωλειών-> Κυρίως λόγο συμπίεσης.

Εικόνα 34: Βίντεο original screenshots.

Εικόνα 35: Βίντεο processed screenshots.

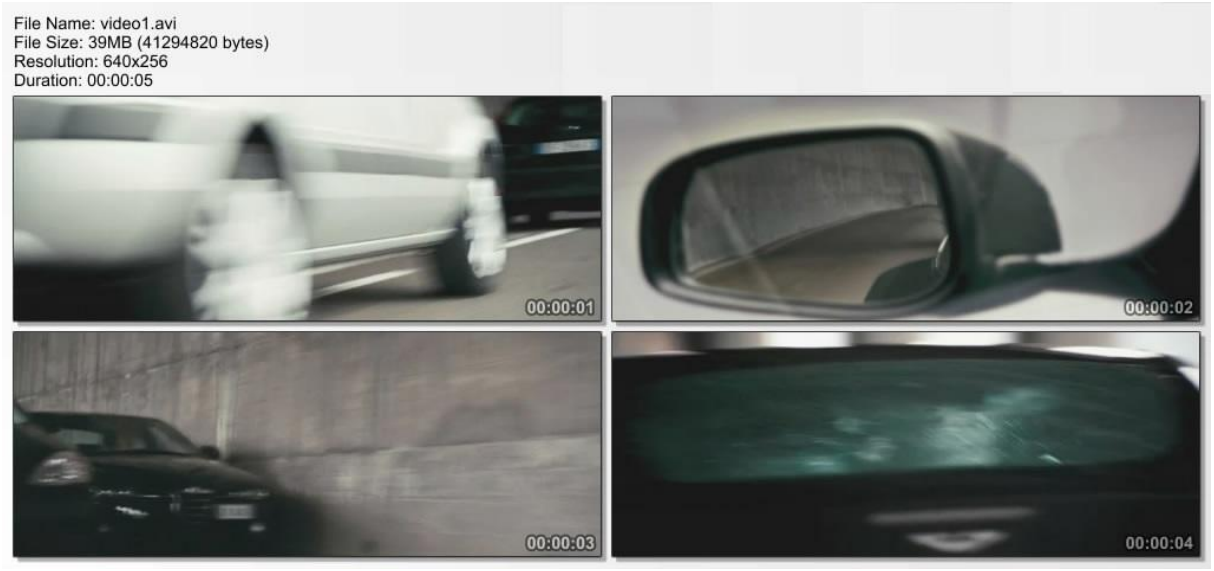
Αριθμός δοκιμής	Τρίτη
Διάρκεια Εκτέλεσης Ανάλυσης	46 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	processed
Μέγεθος αρχείου	3969 Kilo Bytes
Πλάτος	176 pixels
Ύψος	144 pixels
Ρυθμός Καρέ	10 frames/sec
Διάρκεια Αναπαραγωγής	8 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	-
Μέγεθος απώλειας σε bytes	-
Αποτελέσματα Δοκιμής:	
MSE	41,236590
PSNR	31,977976
VQM HV GAIN (μέτρηση Blockiness)	0,026219
VQM HV LOSS (μέτρηση Smearing)	0,219000
VQM SI GAIN (μέτρηση Enchancement)	-0,006923
VQM SI LOSS (μέτρηση Blurring)	0,006651
VQM COHER COLOR (Χρωματικές απώλειες)	0,074704
VQM CONTATI (Απώλειες καρέ)	0,000568
VQM GENERAL	0,123119

Πίνακας 3: Τρίτη δοκιμή.

Σ' αυτή τη δοκιμή, πάλι εξαιτίας λανθασμένης ρύθμισης του κωδικοποιητή, έχει αυξηθεί η φωτεινότητα στο παραμορφωμένο βίντεο αλλά έχει χαθεί η λεπτομέρεια σε κάποια αντικείμενα. Αυτό φαίνεται και στις μετρήσεις (το Coher Color metric είναι 0,0747). Οι τρεις παραπάνω δοκιμές έγιναν για να εξεταστεί ο αλγόριθμος VQM σε περιπτώσεις μικρών απωλειών και απέδειξε ότι προσεγγίζει τις υποκειμενικές εκτιμήσεις.

6.5 Τέταρτη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 20000 bytes, απώλεια 188 bytes.



Εικόνα 36: Βίντεο 1 αρχικό screenshots.

Εικόνα 37: Βίντεο 1 (απώλεια 188 κάθε 20000) screenshots.

Αριθμός δοκιμής	Τέταρτη
Διάρκεια Εκτέλεσης Ανάλυσης	525 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	video 1
Μέγεθος αρχείου	39 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	5 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	20000
Μέγεθος απώλειας σε bytes	188
Αποτελέσματα Δοκιμής:	
MSE	1527,583480
PSNR	16,290754
VQM HV GAIN (μέτρηση Blockiness)	0,229584
VQM HV LOSS (μέτρηση Smearing)	0,395232
VQM SI GAIN (μέτρηση Enchancement)	-0,192864
VQM SI LOSS (μέτρηση Blurring)	0,129376
VQM COHER COLOR (Χρωματικές απώλειες)	0,000000
VQM CONTATI (Απώλειες καρέ)	0,001887
VQM GENERAL	0,563216

Πίνακας 4: Τέταρτη Δοκιμή.

Σ' αυτή τη δοκιμή, έγινε εξομοίωση απωλειών κατά τη μετάδοση, με το πρόγραμμα Add Spaces. Χάθηκαν 188 bytes κάθε 20000 bytes. Το VQM έδωσε χαμηλή βαθμολογία (μεγάλο VQM) επειδή σε κάποια frame η εικόνα έχει παραμορφωθεί εντελώς.

6.6 Πέμπτη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 40000 bytes, απώλεια 376 bytes.

Εικόνα 38: Βίντεο 1 (απώλεια 376 κάθε 40000) screenshots.

Αριθμός δοκιμής	Πέμπτη
Διάρκεια Εκτέλεσης Ανάλυσης	525 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	video 1
Μέγεθος αρχείου	39 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	5 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	40000
Μέγεθος απώλειας σε bytes	376
Αποτελέσματα Δοκιμής:	
MSE	922,135071
PSNR	18,482858
VQM HV GAIN (μέτρηση Blockiness)	0,193405
VQM HV LOSS (μέτρηση Smearing)	0,370691
VQM SI GAIN (μέτρηση Enchancement)	-0,112962
VQM SI LOSS (μέτρηση Blurring)	0,120520
VQM COHER COLOR (Χρωματικές απώλειες)	0,000000
VQM CONTATI (Απώλειες καρέ)	0,002157
VQM GENERAL	0,573810

Πίνακας 5: Πέμπτη δοκιμή.

Στην πέμπτη δοκιμή, οι απώλειες έγιναν πιο αραιά αλλά ήταν μεγαλύτερες. Η παραμόρφωση ήταν παρόμοια με την προηγούμενη δοκιμή γι' αυτό και το VQM παρέμεινε περίπου το ίδιο.

6.7 Έκτη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 40000 bytes, απώλεια 752 bytes.

Εικόνα 39: Βίντεο 1 (απώλεια 752 κάθε 40000) screenshots.

Αριθμός δοκιμής	Έκτη
Διάρκεια Εκτέλεσης Ανάλυσης	526 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	video 1
Μέγεθος αρχείου	38 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	5 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	40000
Μέγεθος απώλειας σε bytes	752
Αποτελέσματα Δοκιμής:	
MSE	3407,694060
PSNR	12,806198
VQM HV GAIN (μέτρηση Blockiness)	0,224047
VQM HV LOSS (μέτρηση Smearing)	0,413036
VQM SI GAIN (μέτρηση Enchancement)	-0,215709
VQM SI LOSS (μέτρηση Blurring)	0,158747
VQM COHER COLOR (Χρωματικές απώλειες)	0,000000
VQM CONTATI (Απώλειες καρέ)	0,001637
VQM GENERAL	0,581759

Πίνακας 6: Έκτη δοκιμή.

Στην έκτη δοκιμή, οι απώλειες μεγάλωσαν όμως η παραμόρφωση και το VQM αυξήθηκαν ελάχιστα. Αυτό συμβαίνει επειδή το βίντεο αποτελείται από γρήγορες σκηνές

(ένα αυτοκίνητο κινείται με μεγάλη ταχύτητα μέσα σε τούνελ) με αποτέλεσμα να χρειάζονται πολλά bytes για την αναπαράστασή του και έτσι οι απώλειες δεν επιδρούν σε μεγάλο βαθμό.

6.8 Έβδομη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 20000 bytes, απώλεια 188 bytes.

Εικόνα 40: Βίντεο 2 αρχικό screenshots.

Εικόνα 41: Βίντεο 2 (απώλεια 188 κάθε 20000) screenshots.

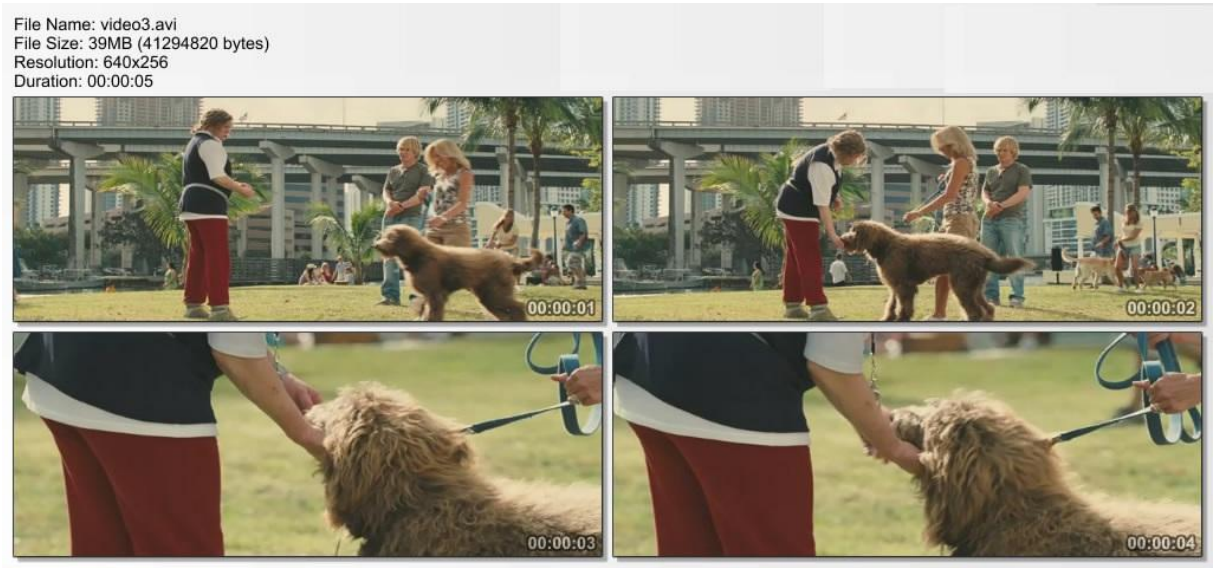
Αριθμός δοκιμής	Έβδομη
Διάρκεια Εκτέλεσης Ανάλυσης	528 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	video 2
Μέγεθος αρχείου	39 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	5 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	20000
Μέγεθος απώλειας σε bytes	188
Αποτελέσματα Δοκιμής:	
MSE	1750,566429
PSNR	15,699018
VQM HV GAIN (μέτρηση Blockiness)	0,225289
VQM HV LOSS (μέτρηση Smearing)	0,441693
VQM SI GAIN (μέτρηση Enchancement)	-0,132423
VQM SI LOSS (μέτρηση Blurring)	0,148565
VQM COHER COLOR (Χρωματικές απώλειες)	0,056474
VQM CONTATI (Απώλειες καρέ)	0,011019
VQM GENERAL	0,750617

Πίνακας 7: Έβδομη δοκιμή.

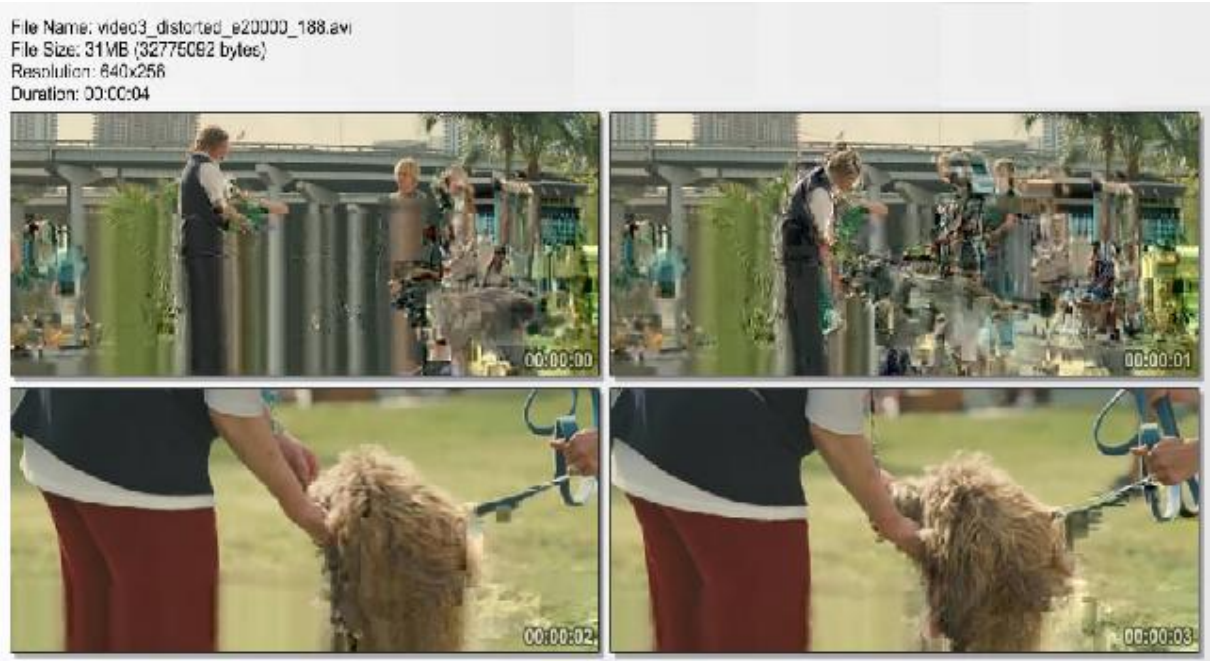
Στην έβδομη δοκιμή, οι απώλειες και το περιεχόμενο ήταν παρόμοια με την τέταρτη δοκιμή αλλά η παραμόρφωση και το VQM ήταν μεγαλύτερα, επειδή εξαρτάται από τα σημεία όπου θα γίνει η απώλεια.

6.9 Όγδοη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 20000 bytes, απώλεια 188 bytes.



Εικόνα 42: Βίντεο 3 αρχικό screenshots.



Εικόνα 43: Βίντεο 3 (απώλεια 188 κάθε 20000) screenshots.

Αριθμός δοκιμής	Όγδοη
Διάρκεια Εκτέλεσης Ανάλυσης	433 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	video 3
Μέγεθος αρχείου	31 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	4 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	20000
Μέγεθος απώλειας σε bytes	188
Αποτελέσματα Δοκιμής:	
MSE	1602,614207
PSNR	16,082514
VQM HV GAIN (μέτρηση Blockiness)	0,306915
VQM HV LOSS (μέτρηση Smearing)	0,508411
VQM SI GAIN (μέτρηση Enchancement)	-0,126566
VQM SI LOSS (μέτρηση Blurring)	0,148588
VQM COHER COLOR (Χρωματικές απώλειες)	0,069974
VQM CONTATI (Απώλειες καρέ)	0,016364
VQM GENERAL	0,923687

Πίνακας 8: Όγδοη δοκιμή.

Στην όγδοη δοκιμή, η παραμόρφωση ήταν μεγάλη και χάθηκαν 8 MB (39 MB ήταν το αρχικό και 31 MB το παραμορφωμένο). Αυτό έγινε, επειδή χάθηκαν οι επικεφαλίδες από το βίντεο και μαζί μ' αυτές το περιεχόμενο στο οποίο ήταν προσαρτημένες. Το βίντεο που κατάφερε να ανακτηθεί από τον αποκωδικοποιητή είχε σφάλματα, με αποτέλεσμα το VQM να είναι μεγάλο (0,923867) και κάποια αντικείμενα στις σκηνές να μην φαίνονται καθόλου. Το περιεχόμενο που χάθηκε δεν εξετάστηκε καθόλου. Παραλείπεται με μία λειτουργία στην αρχή του προγράμματος, για να μπορεί να εξεταστεί το υπόλοιπο. Επειδή τα καρέ του παραμορφωμένου βίντεο, μετά την παράληψη δεν αντιστοιχούνται

με τα καρέ του αρχικού βίντεο, το VQM λειτουργεί προσεγγιστικά, συγκρίνοντας ομάδες από καρέ, για να αντιμετωπίσει αυτό το πρόβλημα.

6.10 Ένατη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 20000 bytes, απώλεια 188 bytes.

Εικόνα 44: Βίντεο 4 αρχικό screenshots.

Εικόνα 45: Βίντεο 4 (απώλεια 188 κάθε 20000) screenshots.

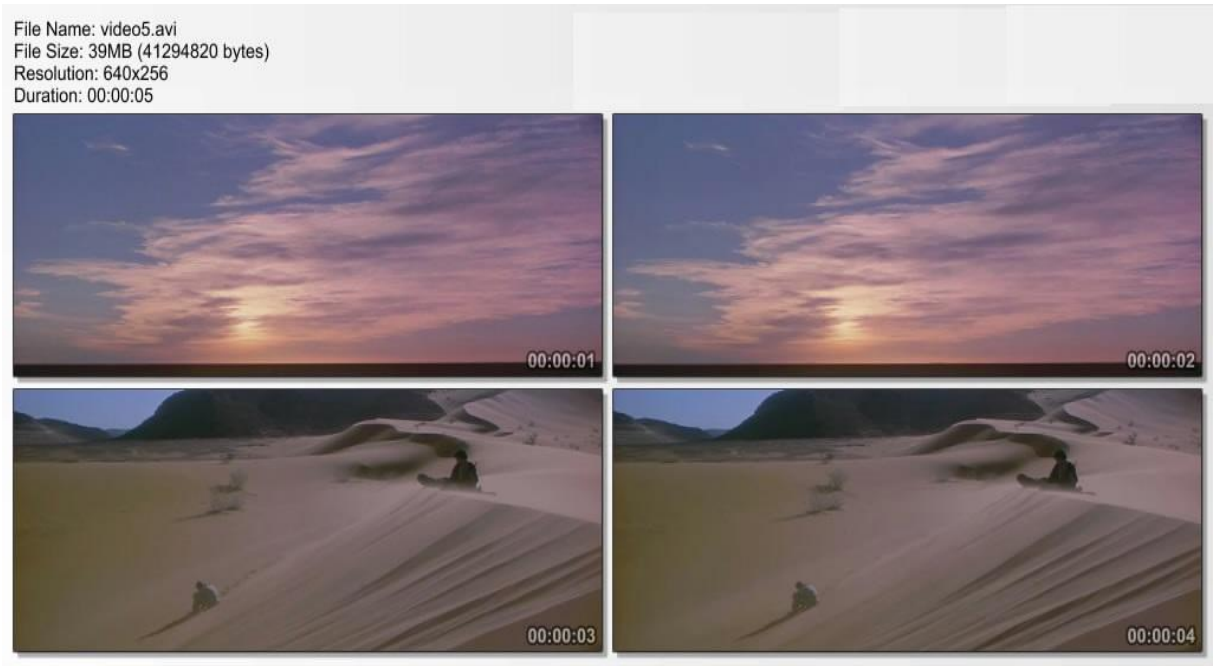
Αριθμός δοκιμής	Ένατη
Διάρκεια Εκτέλεσης Ανάλυσης	539 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	Video 4
Μέγεθος αρχείου	38 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	5 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	20000
Μέγεθος απώλειας σε bytes	188
Αποτελέσματα Δοκιμής:	
MSE	2266,428530
PSNR	14,577383
VQM HV GAIN (μέτρηση Blockiness)	0,285067
VQM HV LOSS (μέτρηση Smearing)	0,524484
VQM SI GAIN (μέτρηση Enchancement)	-0,323584
VQM SI LOSS (μέτρηση Blurring)	0,164237
VQM COHER COLOR (Χρωματικές απώλειες)	0,062358
VQM CONTATI (Απώλειες καρέ)	0,003351
VQM GENERAL	0,715912

Πίνακας 9: Ένατη δοκιμή.

Στην ένατη δοκιμή, το βίντεο περιέχει πολλές πανοραμικές σκηνές με αποτέλεσμα να υποστεί μεγάλη παραμόρφωση από τις απώλειες.

6.11 Δέκατη δοκιμή

Ιδιότητες Βίντεο->Ανάλυση: 640x256, Fps: 25, Διάρκεια: 5 seconds. Ιδιότητες Απωλειών->Κάθε 20000 bytes, απώλεια 188 bytes.



Εικόνα 46: Βίντεο 5 αρχικό screenshots.

Εικόνα 47: Βίντεο 5 (απώλεια 188 κάθε 20000) screenshots.

Αριθμός δοκιμής	Δέκατη
Διάρκεια Εκτέλεσης Ανάλυσης	524 seconds
Χαρακτηριστικά βίντεο:	
Όνομα βίντεο	Video 5
Μέγεθος αρχείου	39 Mega Bytes
Πλάτος	640 pixels
Ύψος	256 pixels
Ρυθμός Καρέ	25 frames/sec
Διάρκεια Αναπαραγωγής	5 seconds
Χαρακτηριστικά απωλειών:	
Κάθε πόσα bytes	20000
Μέγεθος απώλειας σε bytes	188
Αποτελέσματα Δοκιμής:	
MSE	715,480542
PSNR	19,584825
VQM HV GAIN (μέτρηση Blockiness)	0,218321
VQM HV LOSS (μέτρηση Smearing)	0,470500
VQM SI GAIN (μέτρηση Enchancement)	-0,278978
VQM SI LOSS (μέτρηση Blurring)	0,166393
VQM COHER COLOR (Χρωματικές απώλειες)	0,010396
VQM CONTATI (Απώλειες καρέ)	0,000558
VQM GENERAL	0,587190

Πίνακας 10: Δέκατη δοκιμή.

Στην δέκατη δοκιμή, το βίντεο περιέχει επίσης πολλές πανοραμικές σκηνές με αποτέλεσμα να χαθεί μεγάλη ποσότητα πληροφορίας. Και στις δύο δοκιμές παρατηρήθηκε έντονα το φαινόμενο της χρωματικής διασποράς που οδήγησε στην επικάλυψη κάποιων αντικειμένων. Αυτό φαίνεται και από την υψηλή τιμή της μέτρησης smearing.

Παρακάτω φαίνεται πως αντιλαμβάνεται την παραμόρφωση το VQM:

Εικόνα 48: VQM αναπαράσταση.

6.12 Παρατηρήσεις – Συμπεράσματα

Από τις προηγούμενες μετρήσεις επαληθεύεται ότι η ποιότητα ενός βίντεο και κατ' επέκταση η εκτίμηση που έχει πραγματοποιηθεί από τον VQM, όταν έχει υποστεί απώλειες εξαρτάται:

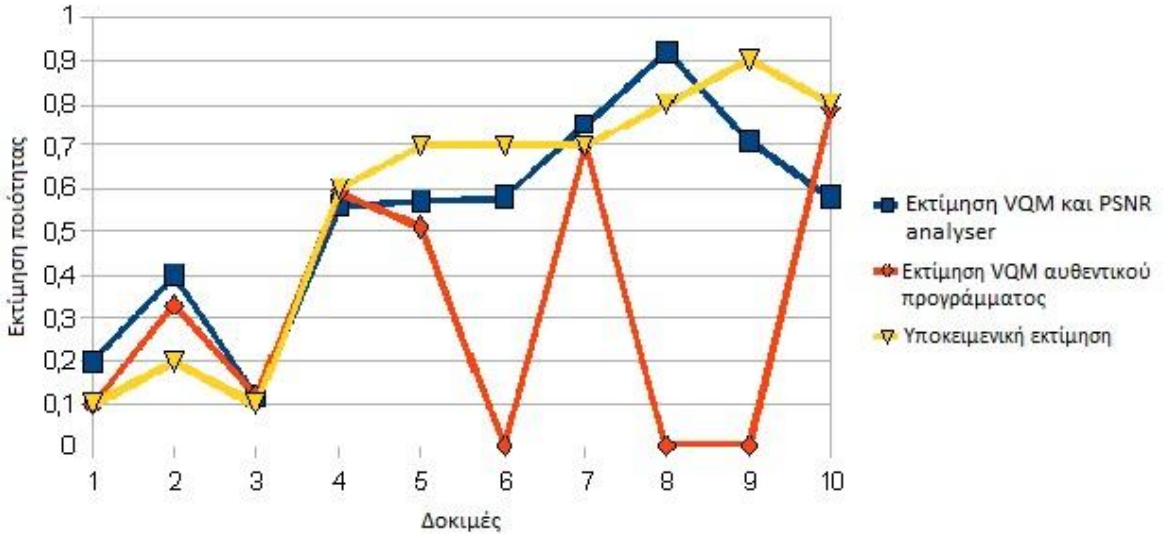
1. Από το bitrate του. Όσο υψηλότερο είναι (που φυσιολογικά αντιστοιχεί σε βίντεο υψηλότερης δυναμικής), τόσο μικρότερη επίδραση (ως ποσοστό) θα έχει η απώλεια ενός block συγκεκριμένου μεγέθους.
2. Από τα σημεία όπου θα υπάρξουν απώλειες. Αν γίνει σε I ή P καρέ θα χαθούν και όλα τα καρέ που αναφέρονται σ' αυτά και έτσι η παραμόρφωση θα είναι μεγάλη.
3. Από το μέγεθος (διάρκεια) και τη συχνότητα της απώλειας, που σε ένα δίκτυο διανομής σχετίζεται με το Byte Error Rate ή/και το Packet Loss Rate
4. Ένα mpeg video όπως αναφέρθηκε στο πρώτο κεφάλαιο εκτός από χρήσιμη πληροφορία περιέχει επικεφαλίδες που δηλώνουν τον τρόπο που είναι αποθηκευμένο. Αν χαθούν οι επικεφαλίδες τότε ο αποκωδικοποιητής δεν θα μπορεί να ανακτήσει κομμάτια του βίντεο καθόλου.
5. Από το είδος του προβλήματος που έχει εμφανιστεί. Πχ. οι απώλειες μεμονωμένων καρέ δεν επηρεάζουν τόσο την ποιότητα όσο το θόλωμα που μπορεί να έχει υποστεί το βίντεο συνολικά.
6. Εξαιτίας της ιδιομορφίας της ανθρώπινης όρασης, η εκτίμηση της ποιότητας ενός βίντεο όταν έχει υποστεί απώλειες εξαρτάται από τη φωτεινότητα και τα χρώματα. Συνεπώς όσο πιο φωτεινό και με έντονα χρώματα είναι το βίντεο, τόσο πιο ορατές είναι οι παραμορφώσεις.
7. Επειδή το VQM μετράει μόνο τις χειρότερες απώλειες, η εκτίμηση δεν εξαρτάται από τη διάρκεια του βίντεο.
8. Εξαρτάται όμως από τον ρυθμό καρέ και την ανάλυση του βίντεο.

Οι δοκιμές πραγματοποιήθηκαν σε 8 βίντεο. Αυτά διέφεραν στην ανάλυση, στον ρυθμό καρέ, στη διάρκεια αλλά και στη χωρική και χρονική πολυπλοκότητα. Στα βίντεο αυτά εκτελέστηκε εξομοίωση μετάδοσης, σε συνθήκες π.χ. ασύρματης μετάδοσης με χαμηλό σηματοθορυβικό λόγο, για την εξέταση της επίδρασης απωλειών στην ποιότητα των βίντεο. Όταν αυξήθηκαν οι απώλειες εμφανίστηκαν πολλές παραμορφώσεις, ειδικά στα βίντεο με αργές και πανοραμικές σκηνές.

Στο παρακάτω γράφημα φαίνονται συγκεντρωτικά τα αποτελέσματα των δοκιμών (VQM metric) με το πρόγραμμα VQM & PSNR analyser. Επίσης παρατίθεται για αναφορά τα αποτελέσματα από τον κώδικα αναφοράς που έχουν δώσει οι σχεδιαστές του αλγορίθμου

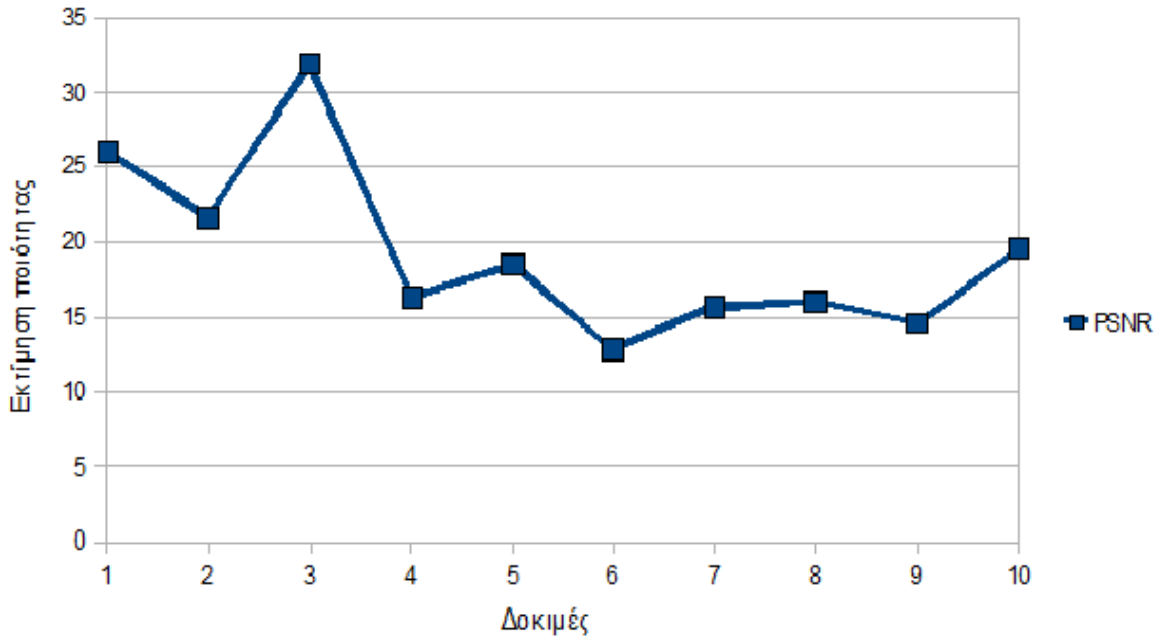
(σε Matlab). Τέλος, έχουν περιληφθεί αποτελέσματα ανθρώπινων εκτιμήσεων(υποκειμενικές εκτιμήσεις). Σε κάθε περίπτωση, η βαθμολόγηση έχει γίνει στην κλίμακα 0-1 όπου το 0 αντιστοιχεί στην απουσία παραμορφώσεων και το 1 σε ένα βίντεο με πολύ έντονες παραμορφώσεις.

Σημειώνεται ότι όπου σημειώνεται VQM score 0 από το πρόγραμμα αναφοράς, αυτό οφείλεται σε αδυναμία εκτέλεσής του λόγω πολύ υψηλών απωλειών στο παραμορφωμένο βίντεο, που έχουν οδηγήσει σε απώλεια της πληροφορίας της εικόνας και καθιστούν αδύνατη την αποκωδικοποίησή του.



Εικόνα 49: Αποτελέσματα δοκιμών VQM.

Παρακάτω φαίνονται συγκεντρωτικά τα αποτελέσματα PSNR.



Εικόνα 50: Αποτελέσματα δοκιμών PSNR.

7 Επίλογος – Μελλοντική ανάπτυξη

Το VQM θεωρείται το καλύτερο metric αυτή τη στιγμή για την εκτίμηση ποιότητας εικόνας και βίντεο. Είναι το πιο αντιπροσωπευτικό όσον αφορά την προσέγγιση της ανθρώπινης εκτίμησης.

Κατά την ανάπτυξή του έγιναν εκατοντάδες δοκιμές σε βίντεο ποικίλου περιεχομένου ώστε να προσεγγίσει τις υποκειμενικές μετρήσεις αλλά και στην αναγνώριση και κατηγοριοποίηση των προβλημάτων που εμφανίζονται συχνά κατά τη συμπίεση και μετάδοση. Τέτοια προβλήματα είναι η εμφάνιση θόλωσης, το φαινόμενο blockiness, η χρωματική διασπορά, οι χρωματικές απώλειες και οι απώλειες καρέ που τις αντιλαμβάνεται χωριστά το metric VQM.

Εκτός όμως από την εκτίμηση βίντεο, μπορεί να χρησιμοποιηθεί και για την εξέταση ενός κωδικοποιητή εικόνας ή ενός συστήματος επικοινωνίας. Μπορεί να συνδυαστεί με τα προαναφερθέντα ώστε να βρεθεί ο βέλτιστος τρόπος συμπίεσης και μετάδοσης. Έτσι θα εξοικονομηθεί εύρος ζώνης συχνοτήτων χωρίς όμως να μειωθεί η ποιότητα της εικόνας που μεταδίδει η υπηρεσία.

8 Βιβλιογραφία

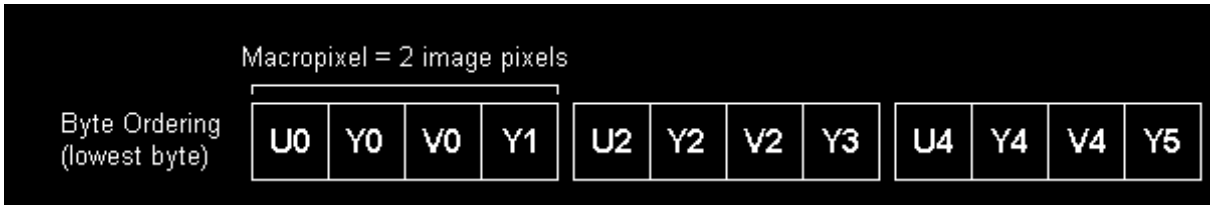
1. Γαρδίκης Γ., "Αμφίδρομη Επίγεια Ψηφιακή Τηλεόραση".
2. Wikipedia, Digital television, "http://en.wikipedia.org/wiki/Digital_television".
3. Γαρδίκης Γ., "Αρχές Ψηφιακής Τηλεόρασης".
4. Κώστας Καρπούζης, "Ψηφιακή επεξεργασία εικόνας".
5. Πολυπλεξία προγραμμάτων με διαμόρφωση QAM, "http://www.sat.gr/show.cfm?id=141&obcatid=22".
6. Wikipedia, Reed–Solomon error correction, "http://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction".
7. Χρήστος Παπακώστας, Μεταπτυχιακή διατριβή στις τεχνολογίες συστημάτων επικοινωνιών & δορυφορικών τηλεπικοινωνιών, Ξάνθη 2004, "Ορθογωνική Πολύπλεξη με Διαίρεση Συχνότητας (OFDM) – Μέθοδοι εκτίμησης του καναλιού μετάδοσης".
8. Wikipedia, Video quality, "http://en.wikipedia.org/wiki/Video_quality".
9. Wikipedia, Peak signal-to-noise ratio, "http://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio".
10. MSU Quality Measurement Tool: Metrics information, "http://compression.ru/video/quality_measure/info_en.html".
11. Wikipedia, Standard deviation, "http://en.wikipedia.org/wiki/Standard_deviation".
12. Steve Wolf, Institute for Telecommunication Sciences, "Measurement of Video Quality".
13. Margaret Pinson, Stephen Wolf, U.S. Department of Commerce, June 2002, NTIA Report 02-392, "Video Quality Measurement Techniques".
14. Sciencepedia, Πολικό Σύστημα Συντεταγμένων, "http://el.science.wikia.com/wiki/Πολικό_Σύστημα".
15. YUV pixel formats, "http://www.fourcc.org/yuv.php".
16. Mplayer, "http://www.mplayerhq.hu/DOCS/HTML/en/index.html".

9 Παράρτημα – Κώδικας, βοηθητικά προγράμματα και πρόσθετες πληροφορίες

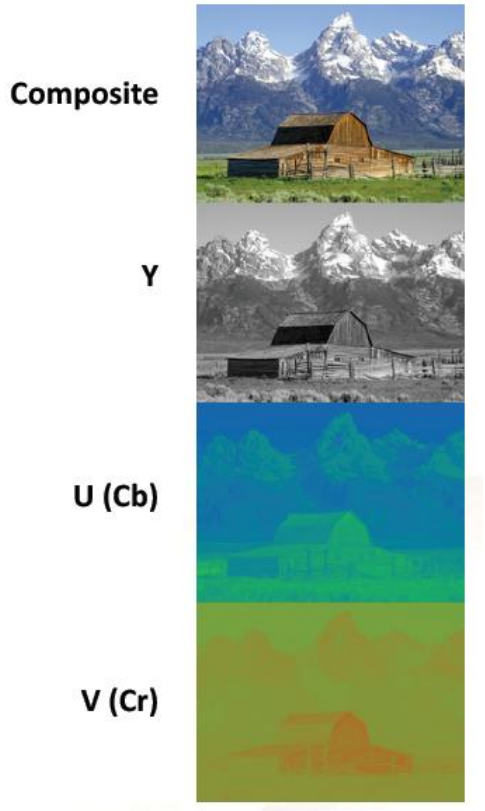
9.1 UYVY Raw format

Έχει τα εξής χαρακτηριστικά:

1. Είναι παραλλαγή του YUV format.
2. Αλλιώς ονομάζεται και Big YUV format.
3. Είναι ασυμπίεστο και δεν είναι αποθηκευμένο μέσα σε container.
4. Τα καρέ του βίντεο αποθηκεύονται σειριακά σε ένα binary αρχείο. Αυτό σημαίνει πως όταν τελειώνουν τα bytes ενός καρέ αρχίζουν τα bytes του επομένου.
5. Κάθε καρέ αποτελείται από γραμμές τόσες όσο το ύψος του βίντεο.
6. Κάθε γραμμή αποτελείται από macropixels τόσα όσο το πλάτος του βίντεο διά δύο.
7. Κάθε macropixel αποτελείται από τέσσερα bytes. Το πρώτο και το τρίτο byte περιέχουν πληροφορία για τη φωτεινότητα ενώ το δεύτερο και το τέταρτο για το μπλε και το κόκκινο χρώμα αντίστοιχα [15]. Όπως:



Εικόνα 51: Αναπαράσταση macropixel.



Εικόνα 52: YUV αναπαράσταση.

Στο επόμενο κεφάλαιο περιγράφω τρόπους πως να μετατραπεί ένα βίντεο σε UYVY format.

9.2 Βοηθητικά προγράμματα

9.2.1 Πρόγραμμα Mplayer

9.2.1.1Περιγραφή

Είναι ένα δωρεάν πρόγραμμα αναπαραγωγής βίντεο για Linux. Μπορεί να παίξει τα περισσότερα format όπως MPEG, VOB, AVI, Ogg/OGM, VIVO, ASF/WMA/WMV, QT/MOV/MP4, FLI, RM, NuppelVideo, yuv4mpeg, FILM, RoQ, PVA, Matroska. Ακόμα υποστηρίζει πολλά output drivers όπως X11, Xv, DGA, OpenGL, SVGAlib, fbdev, AAlib, libcaca, DirectFB, GGI και SDL. Είναι ένας πολύ καλός player για να διαβάζει κατεστραμμένα MPEG ή AVI αρχεία [16].

9.2.1.2Εγκατάσταση

Η διαδικασία είναι απλή αλλά προϋποθέτει να είμαστε συνδεδεμένοι στο Internet.

1. Από το menu του Linux πηγαίνουμε: Applications->Ubuntu Software Center.
2. Κάνουμε αναζήτηση το mplayer.
3. Κάνουμε click στο Install.

9.2.1.3Χρήση

Όταν το video περιέχει container η διαδικασία είναι απλή:

1. Τρέχουμε το MPlayer Media Player που βρίσκεται στο menu: Applications->Sound & Video
2. Σύρουμε το video μέσα στο παράθυρο και αρχίζει η αναπαραγωγή.

Όταν το video δεν έχει container πρέπει να δώσουμε κάποιες επιπλέον παραμέτρους από το Terminal για να λειτουργήσει σωστά. Παρακάτω περιγράφω κάποιες βασικές παραμέτρους:

- demuxer rawvideo: Δηλώνω στον αποκωδικοποιητή ότι το βίντεο είναι ασυμπίεστης μορφής.
- rawvideo w=640:h=256:fps=25:format=uyvy: Δηλώνω ότι το βίντεο έχει πλάτος 640, ύψος 256, ρυθμό καρέ 25 και format UYVY
- zoom: Δίνει τη δυνατότητα να δούμε το βίντεο σε full screen.

Η πλήρης εντολή για να δούμε ένα βίντεο UYVY είναι η παρακάτω:

gmplayer '/home/sarafo/Desktop/videos/video1.yuv' -demuxer rawvideo -rawvideo w=640:h=256:fps=25:format=uyvy -zoom

9.2.2 Πρόγραμμα Mencoder

9.2.2.1Περιγραφή

Ο mencoder είναι ένας απλός movie encoder, σχεδιασμένος να κωδικοποιεί βίντεο που παίζουν στον mplayer όπως AVI/ ASF/ OGG/ DVD/ VCD/ VOB/ MPG/ MOV/ VIV/ FLI/ RM/ NUV/ NET/ PVA σε άλλα βίντεο που παίζουν στον mplayer. Μπορεί να κωδικοποιήσει σε πολλά codec όπως divx, libavcodec, PCM/MP3/VBR MP3 audio.

9.2.2.2Εγκατάσταση

1. Από το menu του Linux: System->Administration->Synaptic Package Manager
2. Κάνουμε αναζήτηση τον mencoder.
3. Τον επιλέγουμε και πατάμε apply.

9.2.2.3Χρήση

Είναι command-line πρόγραμμα, γι' αυτό ανοίγουμε το Terminal. Παρακάτω περιγράφονται κάποιες βασικές παραμέτρους που χρειάζονται για να κωδικοποιήσουμε UYVY RAW βίντεο:

- ovc raw: Δηλώνει ότι το format του βίντεο θα είναι ασυμπίεστο.
- vf format=uyvy: Δηλώνει ότι το format του βίντεο θα είναι UYVY.
- of raw: Δηλώνει ότι δεν θα εμπεριέχεται σε container.

Η πλήρης εντολή για να μετατρέψουμε ένα βίντεο σε UYVY RAW είναι:

mencoder '/home/sarafo/Desktop/mp4 videos/video_1.mp4' -ovc raw -vf format=uyvy -of raw -o video_1.yuv

Όταν το βίντεο περιέχει ήχο προσθέτουμε την παράμετρο -nosound και αφαιρείται τελείως.

9.3 Πηγαίος κώδικας του προγράμματος VQM & PSNR analyser

```
//compareVideos made by SrF -- Sarandou Fotios
//////////////////////
// *** *** *** //
// ** * * * //
// ** *** *** //
// ** ** * //
// *** * * * //
//////////////////////
#include <iostream>
#include <fstream>
#include <cmath>
#include <iomanip>
#include <ctime>
#include <cstdlib>
using namespace std;
//variables
long long unsigned int
size=0,size2=0,myBuffer=0,width=0,height=0,oneFrame=0,framesOnBuffer=0;
long long unsigned int x=0,y=0,t=0,i=0,j=0,l=0;
long long unsigned int startt=0,stopt=0,startx=0,starty=0,stopy=0,stopx=0;
long double psnr=0,vqm=0;
double temp=0;
unsigned char orig=0,proc=0;
time_t rawtime=0,rawtime2=0;
char * memblock;
char * memblock2;
int main (int argc, char *argv[]) {
cout<<endl<<"You are running VQM & PSNR analyser made by SrF"<<endl;
//open files
if (argc!=5)
{
cout<<"run program with these arguments <compareVideos.exe \"file1.x\" \"file2.x\" width
height>"<<endl;
return 0;
}
if ((string)argv[1]==(string)argv[2])
{
```

```
cout<<"you choose the same file twice"<<endl;
return 0;
}
ifstream file (argv[1],ios::in|ios::binary|ios::ate);
if (file.is_open()==0)
{
cout<<"there was a problem, opening first file"<<endl;
return 0;
}
ifstream file2 (argv[2],ios::in|ios::binary|ios::ate);
if (file2.is_open()==0)
{
cout<<"there was a problem, opening second file"<<endl;
return 0;
}
//check if the files have the right size
size = file.tellg();
size2 = file2.tellg();
if (size<size2)
{
cout<<"original can't be smaller than processed"<<endl;
return 0;
}
//buffering and processing
size=size2;
myBuffer=size;
width=atoi(argv[3])*2;
height=atoi(argv[4]);
oneFrame=width*height;
framesOnBuffer=myBuffer/oneFrame;
if(((oneFrame*framesOnBuffer)!=size) || ((width%16)!=0) || ((height%8)!=0))
{
cout<<"wrong width or height"<<endl;
return 0;
}
time(&rawtime);
cout<<"program started at "<<ctime(&rawtime);
cout<<"please wait..."<<endl;
memblock = new char[myBuffer];
memblock2 = new char[myBuffer];
file.seekg(ios::beg);
file.read (memblock,myBuffer);
file2.seekg(ios::beg);
file2.read (memblock2,myBuffer);
file.close();
file2.close();
//variables COLOR MSE
long long unsigned int maxiC=size/128;
long long unsigned int maxiCFrame=oneFrame/64;
double color10per=0;
long long unsigned int framesOnBuffer10per=framesOnBuffer/10;
long double mse=0;
double * colorEuclid;
double * meanColorOrigCb;
double * meanColorOrigCr;
```

```
double * meanColorProcCb;
double * meanColorProcCr;
double * stdColor;
double * stdMeanColor;
//create COLOR
colorEuclid = new double[size/64];
meanColorOrigCb = new double[maxiC];
meanColorOrigCr = new double[maxiC];
meanColorProcCb = new double[maxiC];
meanColorProcCr = new double[maxiC];
stdColor = new double[maxiCFrame];
stdMeanColor = new double[maxiCFrame];
//COHER COLOR & MSE
maxiC=0;
while(t<framesOnBuffer)
{
startx=0;
stopx=16;
stopy=8;
x=0;
y=0;
while(x<width)
{
while(y<height)
{
meanColorOrigCb[maxiC]=0;
meanColorOrigCr[maxiC]=0;
meanColorProcCb[maxiC]=0;
meanColorProcCr[maxiC]=0;
while(y<stopy)
{
while(x<stopx)
{
//color U
orig=memblock[x+(y*width)+(t*oneFrame)];
proc=memblock2[x+(y*width)+(t*oneFrame)];
//COHER COLOR
meanColorOrigCb[maxiC]+=orig;
meanColorProcCb[maxiC]+=proc;
x++; //color Y
orig=memblock[x+(y*width)+(t*oneFrame)];
proc=memblock2[x+(y*width)+(t*oneFrame)];
//MSE
mse+=((orig-proc)*(orig-proc));
x++; //color V
orig=memblock[x+(y*width)+(t*oneFrame)];
proc=memblock2[x+(y*width)+(t*oneFrame)];
//COHER COLOR
meanColorOrigCr[maxiC]+=orig;
meanColorProcCr[maxiC]+=proc;
x++; //color Y
orig=memblock[x+(y*width)+(t*oneFrame)];
proc=memblock2[x+(y*width)+(t*oneFrame)];
//MSE
mse+=((orig-proc)*(orig-proc));
```

```
x++; //next macropixel
}
x=startx;
y++;
}
stopy+=8;
meanColorOrigCb[maxiC]/=32;
meanColorOrigCr[maxiC]=(meanColorOrigCr[maxiC]/32)*1.5;
meanColorProcCb[maxiC]/=32;
meanColorProcCr[maxiC]=(meanColorProcCr[maxiC]/32)*1.5;
maxiC++;
}
startx+=16;
stopx+=16;
x=startx;
stopy=8;
y=0;
}
t++;
}
//COHER COLOR
//euclid COLOR
maxiC*=2;
i=0;
j=0;
while(i<maxiC)
{
colorEuclid[i]=sqrt((meanColorOrigCb[j]-meanColorProcCb[j])*(meanColorOrigCb[j]-
meanColorProcCb
[j]));
i++;
colorEuclid[i]=sqrt((meanColorOrigCr[j]-meanColorProcCr[j])*(meanColorOrigCr[j]-
meanColorProcCr
[j]));
i++;
j++;
}
//mean std COLOR
t=0;
i=0;
j=0;
while(t<maxiC)
{
stdMeanColor[j]=0;
while(i<(t+maxiCFrame))
{
stdMeanColor[j]+=colorEuclid[i];
i++;
}
t+=maxiCFrame;
stdMeanColor[j]/=maxiCFrame;
j++;
}
//std COLOR
t=0;
```

```
i=0;
j=0;
while(t<maxiC)
{
stdColor[j]=0;
while(i<(t+maxiCFrame))
{
stdColor[j]+=(colorEuclid[i]-stdMeanColor[j])*(colorEuclid[i]-stdMeanColor[j]);
i++;
}
t+=maxiCFrame;
stdColor[j]=sqrt(stdColor[j]/maxiCFrame);
j++;
}
//sort 10% percent COLOR
for(i=0; i<=framesOnBuffer10per; i++)
{
for(j=i; j<framesOnBuffer; j++)
{
if(stdColor[j]<stdColor[i])
{
temp=stdColor[i];
stdColor[i]=stdColor[j];
stdColor[j]=temp;
}
}
}
color10per=stdColor[framesOnBuffer10per];
//clean COLOR
delete[] stdColor;
delete[] stdMeanColor;
delete[] meanColorOrigCr;
delete[] meanColorOrigCb;
delete[] meanColorProcCr;
delete[] meanColorProcCb;
delete[] colorEuclid;
//variables CONTATI
long long unsigned int maxiY=size/192;
long double
meanContOrig=0,meanContProc=0,meanAtiOrig=0,meanAtiProc=0,contatiOrig=0,contatiProc=0;
long long unsigned int contatiRegions=(oneFrame*6)/192;
double contati10per=0;
long long unsigned int maxiY10per=0;
unsigned char prevOrig=0,prevProc=0;
double * stdContOrig;
double * stdContProc;
double * stdAtiOrig;
double * stdAtiProc;
double * contati;
double * meanContati;
//create CONTATI
stdContOrig = new double[maxiY];
stdContProc = new double[maxiY];
stdAtiOrig = new double[maxiY];
stdAtiProc = new double[maxiY];
```

```
contati = new double[maxiY];
meanContati = new double[maxiY/contatiRegions];
//CONTATI MEAN
maxiY=0;
i=80;
startt=1;
t=startt;
stopt=6;
while(stopt<framesOnBuffer)
{
startx=0;
starty=0;
stopx=8;
stopy=4;
x=startx;
y=starty;
while(x<width)
{
while(y<height)
{
stdContOrig[maxiY]=0;
stdContProc[maxiY]=0;
stdAtiOrig[maxiY]=0;
stdAtiProc[maxiY]=0;
while(t<stopt)
{
while(y<stopy)
{
while(x<stopx)
{
x++; //color Y
orig=memblock[x+(y*width)+(t*oneFrame)];
proc=memblock2[x+(y*width)+(t*oneFrame)];
stdContOrig[maxiY]+=orig;
stdContProc[maxiY]+=proc;
prevOrig=memblock[x+(y*width)+((t-1)*oneFrame)];
prevProc=memblock2[x+(y*width)+((t-1)*oneFrame)];
stdAtiOrig[maxiY]+=sqrt((orig-prevOrig)*(orig-prevOrig));
stdAtiProc[maxiY]+=sqrt((proc-prevProc)*(proc-prevProc));
x++; //next macropixel
}
x=startx;
y++;
}
t++;
x=startx;
y=starty;
}
t=startt;
starty+=4;
y=starty;
stopy+=4;
stdContOrig[maxiY]/=i;
stdContProc[maxiY]/=i;
stdAtiOrig[maxiY]/=i;
```

```
stdAtiProc[maxiY]/=i;
maxiY++;
}
startx+=8;
stopx+=8;
x=startx;
starty=0;
stopy=4;
y=starty;
}
if(startt==1)
{
startt+=5;
i=96;
}
else
{
startt+=6;
}
t=startt;
stopt+=6;
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<fixed<<setprecision(0)<<(((double)(t*oneFrame)/size)*15)<<"%";
}
//CONTATI STD
maxiY=0;
i=80;
startt=1;
t=startt;
stopt=6;
while(stopt<framesOnBuffer)
{
startx=0;
starty=0;
stopx=8;
stopy=4;
x=startx;
y=starty;
while(x<width)
{
while(y<height)
{
meanContOrig=stdContOrig[maxiY];
meanContProc=stdContProc[maxiY];
meanAtiOrig=stdAtiOrig[maxiY];
meanAtiProc=stdAtiProc[maxiY];
stdContOrig[maxiY]=0;
stdContProc[maxiY]=0;
stdAtiOrig[maxiY]=0;
stdAtiProc[maxiY]=0;
while(t<stopt)
```

```
{
while(y<stopy)
{
while(x<stopx)
{
x++; //color Y
orig=memblock[x+(y*width)+(t*oneFrame)];
proc=memblock2[x+(y*width)+(t*oneFrame)];
stdContOrig[maxiY]+=(orig-meanContOrig)*(orig-meanContOrig);
stdContProc[maxiY]+=(proc-meanContProc)*(proc-meanContProc);
prevOrig=memblock[x+(y*width)+((t-1)*oneFrame)];
prevProc=memblock2[x+(y*width)+((t-1)*oneFrame)];
stdAtiOrig[maxiY]+=(sqrt((orig-prevOrig)*(orig-prevOrig))-meanAtiOrig)*(sqrt
((orig-prevOrig)*(orig-prevOrig))-meanAtiOrig);
stdAtiProc[maxiY]+=(sqrt((proc-prevProc)*(proc-prevProc))-meanAtiProc)*(sqrt
((proc-prevProc)*(proc-prevProc))-meanAtiProc);
x++; //next macropixel
}
x=startx;
y++;
}
t++;
x=startx;
y=starty;
}
t=startt;
starty+=4;
y=starty;
stopy+=4;
stdContOrig[maxiY]=sqrt(stdContOrig[maxiY]/i);
stdContProc[maxiY]=sqrt(stdContProc[maxiY]/i);
stdAtiOrig[maxiY]=sqrt(stdAtiOrig[maxiY]/i);
stdAtiProc[maxiY]=sqrt(stdAtiProc[maxiY]/i);
maxiY++;
}
startx+=8;
stopx+=8;
x=startx;
starty=0;
stopy=4;
y=starty;
}
if(startt==1)
{
startt+=5;
i=96;
}
else
{
startt+=6;
}
t=startt;
stopt+=6;
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
```

```
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<(((double)(t*oneFrame)/size)*15)+15<<"%";
}
//CONTATI
//ratio gain CONTATI
for(i=0; i<maxiY; i++)
{
if(stdContOrig[i]<3)
stdContOrig[i]=3;
if(stdContProc[i]<3)
stdContProc[i]=3;
if(stdAtiOrig[i]<3)
stdAtiOrig[i]=3;
if(stdAtiProc[i]<3)
stdAtiProc[i]=3;
contatiOrig=stdContOrig[i]*stdAtiOrig[i];
contatiProc=stdContProc[i]*stdAtiProc[i];
contati[i]=0;
if(contatiProc>contatiOrig)
contati[i]=(contatiProc-contatiOrig)/contatiOrig;
}
//mean CONTATI
t=0;
for(i=0; i<maxiY; i+=contatiRegions)
{
meanContati[t]=0;
for (j=0; j<contatiRegions; j++)
{
meanContati[t]+=contati[i+j];
}
meanContati[t]/=contatiRegions;
t++;
}
maxiY=t;
//sort 10% CONTATI
maxiY10per=maxiY*0.1;
for(i=0; i<=maxiY10per; i++)
{
for(j=i; j<maxiY; j++)
{
if(meanContati[j]<meanContati[i])
{
temp=meanContati[i];
meanContati[i]=meanContati[j];
meanContati[j]=temp;
}
}
}
contati10per=meanContati[maxiY10per];
//clean CONTATI
delete[] stdContOrig;
delete[] stdContProc;
delete[] stdAtiOrig;
```

```
delete[] stdAtiProc;
delete[] contati;
delete[] meanContati;
//variables HV SI
double hOrig=0,vOrig=0,hProc=0,vProc=0,rOrig=0,rProc=0,thOrig=0,thProc=0;
double hv13Orig=0,hv13Proc=0;
double hv13=0,hvLoss=0,logSI=0,lossSI=0;
double meanHVBelow=0,meanSI=0,siMean5per=0,meanHV=0,meanHV5per=0;
double hvMeanOrig=0,hvMeanProc=0,hv2MeanOrig=0,hv2MeanProc=0;
long long unsigned int maxiHV5per=0,maxiSI5per=0,maxiHVGain=0,maxiHVLoss=0,maxiSI=0;
long long unsigned int maxiHVGainCount=0,maxiHVLossCount=0;
long long unsigned int hvOneFrame=(width-32)*(height-16);
long long unsigned int hvSize=(hvOneFrame*framesOnBuffer)/2;
long long unsigned int maxiHV=hvSize/384;
long long unsigned int maxiHVOneFrame=hvOneFrame/128;
int k=0;
double * siMeanOrig;
double * siMeanProc;
double * siOrig;
double * siProc;
double * logHV;
double * ratioHV;
double * stdSIOrig;
double * stdSIProc;
double * ratioSI;
//create HV SI
siMeanOrig = new double[maxiHV];
siMeanProc = new double[maxiHV];
siOrig = new double[hvSize];
siProc = new double[hvSize];
logHV = new double[maxiHVOneFrame];
ratioHV = new double[maxiHVOneFrame];
//SI & HV
width-=16;
height-=8;
maxiHV=0;
startt=0;
t=startt;
stopt=6;
i=0;
while(stopt<framesOnBuffer)
{
maxiHVGain=0;
maxiHVLoss=0;
startx=16;
starty=8;
stopx=32;
stopy=16;
x=startx;
y=starty;
while(x<width)
{
while(y<height)
{
siMeanOrig[maxiHV]=0;
```

```
siMeanProc[maxiHV]=0;
hvMeanOrig=0;
hvMeanProc=0;
hv2MeanOrig=0;
hv2MeanProc=0;
while(t<stopt)
{
while(y<stopy)
{
while(x<stopx)
{
x++; //color Y
//horizontal filter
hOrig=0;
hProc=0;
for(k=-6; k<=6; k++)
{
orig=memblock[(x-12)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x-12)+((y+k)*width)+(t*oneFrame)];
hOrig-=orig*0.0052625;
hProc-=proc*0.0052625;
orig=memblock[(x-10)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x-10)+((y+k)*width)+(t*oneFrame)];
hOrig-=orig*0.0173446;
hProc-=proc*0.0173446;
orig=memblock[(x-8)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x-8)+((y+k)*width)+(t*oneFrame)];
hOrig-=orig*0.0427401;
hProc-=proc*0.0427401;
orig=memblock[(x-6)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x-6)+((y+k)*width)+(t*oneFrame)];
hOrig-=orig*0.0768961;
hProc-=proc*0.0768961;
orig=memblock[(x-4)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x-4)+((y+k)*width)+(t*oneFrame)];
hOrig-=orig*0.0957739;
hProc-=proc*0.0957739;
orig=memblock[(x-2)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x-2)+((y+k)*width)+(t*oneFrame)];
hOrig-=orig*0.0696751;
hProc-=proc*0.0696751;
orig=memblock[(x+2)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x+2)+((y+k)*width)+(t*oneFrame)];
hOrig+=orig*0.0696751;
hProc+=proc*0.0696751;
orig=memblock[(x+4)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x+4)+((y+k)*width)+(t*oneFrame)];
hOrig+=orig*0.0957739;
hProc+=proc*0.0957739;
orig=memblock[(x+6)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x+6)+((y+k)*width)+(t*oneFrame)];
hOrig+=orig*0.0768961;
hProc+=proc*0.0768961;
orig=memblock[(x+8)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x+8)+((y+k)*width)+(t*oneFrame)];
```

```
hOrig+=orig*0.0427401;
hProc+=proc*0.0427401;
orig=memblock[(x+10)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x+10)+((y+k)*width)+(t*oneFrame)];
hOrig+=orig*0.0173446;
hProc+=proc*0.0173446;
orig=memblock[(x+12)+((y+k)*width)+(t*oneFrame)];
proc=memblock2[(x+12)+((y+k)*width)+(t*oneFrame)];
hOrig+=orig*0.0052625;
hProc+=proc*0.0052625;
}
//vertical filter
vOrig=0;
vProc=0;
for(k=-12; k<=12; k+=2)
{
orig=memblock[(x+k)+((y-6)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y-6)*width)+(t*oneFrame)];
vOrig-=orig*0.0052625;
vProc-=proc*0.0052625;
orig=memblock[(x+k)+((y-5)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y-5)*width)+(t*oneFrame)];
vOrig-=orig*0.0173446;
vProc-=proc*0.0173446;
orig=memblock[(x+k)+((y-4)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y-4)*width)+(t*oneFrame)];
vOrig-=orig*0.0427401;
vProc-=proc*0.0427401;
orig=memblock[(x+k)+((y-3)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y-3)*width)+(t*oneFrame)];
vOrig-=orig*0.0768961;
vProc-=proc*0.0768961;
orig=memblock[(x+k)+((y-2)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y-2)*width)+(t*oneFrame)];
vOrig-=orig*0.0957739;
vProc-=proc*0.0957739;
orig=memblock[(x+k)+((y-1)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y-1)*width)+(t*oneFrame)];
vOrig-=orig*0.0696751;
vProc-=proc*0.0696751;
orig=memblock[(x+k)+((y+1)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y+1)*width)+(t*oneFrame)];
vOrig+=orig*0.0696751;
vProc+=proc*0.0696751;
orig=memblock[(x+k)+((y+2)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y+2)*width)+(t*oneFrame)];
vOrig+=orig*0.0957739;
vProc+=proc*0.0957739;
orig=memblock[(x+k)+((y+3)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y+3)*width)+(t*oneFrame)];
vOrig+=orig*0.0768961;
vProc+=proc*0.0768961;
orig=memblock[(x+k)+((y+4)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y+4)*width)+(t*oneFrame)];
vOrig+=orig*0.0427401;
```

```
vProc+=proc*0.0427401;
orig=memblock[(x+k)+((y+5)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y+5)*width)+(t*oneFrame)];
vOrig+=orig*0.0173446;
vProc+=proc*0.0173446;
orig=memblock[(x+k)+((y+6)*width)+(t*oneFrame)];
proc=memblock2[(x+k)+((y+6)*width)+(t*oneFrame)];
vOrig+=orig*0.0052625;
vProc+=proc*0.0052625;
}
rOrig=sqrt((hOrig*hOrig)+(vOrig*vOrig));
rProc=sqrt((hProc*hProc)+(vProc*vProc));
thOrig=atan(vOrig/hOrig);
thProc=atan(vProc/hProc);
siMeanOrig[maxiHV]+=rOrig;
siMeanProc[maxiHV]+=rProc;
siOrig[i]=rOrig;
siProc[i]=rProc;
i++;
if((rOrig>=20)&&(((-0.225<thOrig)&&(thOrig<0.225))||((1.3457<thOrig)&&
(thOrig<1.7957))||((2.9165<thOrig)&&(thOrig<3.3665))||((4.4873<thOrig)&&(thOrig<4.9373))))
{
hvMeanOrig+=rOrig;
}
if((rOrig>=20)&&(((0.225<=thOrig)&&(thOrig<=1.3457))||((1.7957<=thOrig)&&
(thOrig<=2.9165))||((3.3665<=thOrig)&&(thOrig<=4.4873))||((4.9373<=thOrig)&&(thOrig<=6.0581)
)))
{
hv2MeanOrig+=rOrig;
}
if((rProc>=20)&&(((-0.225<thProc)&&(thProc<0.225))||((1.3457<thProc)&&
(thProc<1.7957))||((2.9165<thProc)&&(thProc<3.3665))||((4.4873<thProc)&&(thProc<4.9373))))
{
hvMeanProc+=rProc;
}
if((rProc>=20)&&(((0.225<=thProc)&&(thProc<=1.3457))||((1.7957<=thProc)&&
(thProc<=2.9165))||((3.3665<=thProc)&&(thProc<=4.4873))||((4.9373<=thProc)&&(thProc<=6.0581)
)))
{
hv2MeanProc+=rProc;
}
x++; //next macropixel
}
x=startx;
y++;
}
t++;
x=startx;
y=starty;
}
t=startt;
starty+=8;
y=starty;
stopy+=8;
siMeanOrig[maxiHV]/=384;
```

```
siMeanProc[maxiHV]/=384;
hvMeanOrig/=384;
hvMeanProc/=384;
hv2MeanOrig/=384;
hv2MeanProc/=384;
if(hvMeanOrig<3)
hvMeanOrig=3;
if(hv2MeanOrig<3)
hv2MeanOrig=3;
if(hvMeanProc<3)
hvMeanProc=3;
if(hv2MeanProc<3)
hv2MeanProc=3;
hv13Orig=hvMeanOrig/hv2MeanOrig;
hv13Proc=hvMeanProc/hv2MeanProc;
//log HV
hv13=log10(hv13Proc/hv13Orig);
if(hv13>0)
{
logHV[maxiHVGain]=hv13;
maxiHVGain++;
}
//ratio HV
hvLoss=(hv13Proc-hv13Orig)/hv13Orig;
if (hvLoss<0)
{
ratioHV[maxiHVLoss]=hvLoss;
maxiHVLoss++;
}
maxiHV++;
}
startx+=16;
stopx+=16;
x=startx;
starty=8;
stopy=16;
y=starty;
}
//sort HV
maxiHV5per=0;
meanHV5per=0;
if(maxiHVGain>0)
{
maxiHV5per=maxiHVGain*0.05;
maxiHV5per++;
}
for (j=0; j<=maxiHV5per; j++)
{
for (l=j; l<maxiHVGain; l++)
{
if(logHV[l]>logHV[j])
{
temp=logHV[j];
logHV[j]=logHV[l];
logHV[l]=temp;
```

```
}
}
meanHV5per+=logHV[j];
}
if(maxiHV5per>0)
{
meanHV5per/=(maxiHV5per+1);
meanHV+=meanHV5per;
maxiHVGainCount++;
}
//sort HV below
maxiHV5per=0;
meanHV5per=0;
if(maxiHVLoss>0)
{
maxiHV5per=maxiHVLoss*0.05;
maxiHV5per++;
}
for (j=0; j<=maxiHV5per; j++)
{
for (l=j; l<maxiHVLoss; l++)
{
if(ratioHV[l]<ratioHV[j])
{
temp=ratioHV[j];
ratioHV[j]=ratioHV[l];
ratioHV[l]=temp;
}
}
meanHV5per+=ratioHV[j];
}
if(maxiHV5per>0)
{
meanHV5per/=(maxiHV5per+1);
meanHVBelow+=meanHV5per;
maxiHVLossCount++;
}
startt+=6;
t=startt;
stopt+=6;
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<(((double)(t*oneFrame)/size)*55)+30<<"%";
```

```
}
//clean buffer
delete[] memblock;
delete[] memblock2;
delete[] logHV;
delete[] ratioHV;
//create SI
stdSIOrig = new double[maxiHV];
stdSIProc = new double[maxiHV];
//SI STD
for(i=0; i<maxiHV; i++)
{
stdSIOrig[i]=0;
stdSIProc[i]=0;
for(j=0; j<384; j++)
{
stdSIOrig[i]+=(siOrig[j+(i*384)]-siMeanOrig[i])*(siOrig[j+(i*384)]-siMeanOrig[i]);
stdSIProc[i]+=(siProc[j+(i*384)]-siMeanProc[i])*(siProc[j+(i*384)]-siMeanProc[i]);
}
stdSIOrig[i]=sqrt(stdSIOrig[i]/384);
stdSIProc[i]=sqrt(stdSIProc[i]/384);
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<(((double)i/maxiHV)*11)+85<<"%";
}
//clean SI
delete[] siMeanOrig;
delete[] siMeanProc;
delete[] siOrig;
delete[] siProc;
//create SI
ratioSI = new double[maxiHV];
//SI
//log & mean SI
for(i=0; i<maxiHV; i++)
{
if(stdSIOrig[i]<8)
stdSIOrig[i]=8;
if(stdSIProc[i]<8)
stdSIProc[i]=8;
logSI=log10(stdSIProc[i]/stdSIOrig[i]);
if(logSI>0)
{
meanSI+=logSI;
}
}
meanSI/=maxiHV;
//percent SI
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
```

```
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<"97%";
//ratio loss SI
for(i=0; i<maxiHV; i++)
{
ratioSI[i]=0;
if(stdSIOrig[i]<12)
stdSIOrig[i]=12;
if(stdSIProc[i]<12)
stdSIProc[i]=12;
lossSI=(stdSIProc[i]-stdSIOrig[i])/stdSIOrig[i];
if(lossSI<0)
{
ratioSI[maxiSI]=lossSI;
maxiSI++;
}
}
//percent SI
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<"98%";
//below 5% mean SI
maxiSI5per=maxiHV*0.05;
for(i=0; i<maxiSI5per; i++)
{
for(j=i; j<maxiSI; j++)
{
if(ratioSI[j]<ratioSI[i])
{
temp=ratioSI[i];
ratioSI[i]=ratioSI[j];
ratioSI[j]=temp;
}
}
siMean5per+=ratioSI[i];
}
siMean5per/=maxiSI5per;
//clean SI
delete[] ratioSI;
//percent SI
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<"99%";
//HV
//mean HV
if(maxiHVGainCount>0)
meanHV/=maxiHVGainCount;
//below mean HV
if(maxiHVLossCount>0)
```

```
{
meanHVBelow/=maxiHVLossCount;
meanHVBelow*=meanHVBelow;
}
//percent HV
cout<<"\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b
\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b";
cout<<"100%";
//results
cout<<endl<<"the size of files is "<<size<<endl;
if (mse==0)
{
cout<<"videos are the same"<<endl;
return 0;
}
mse/=(size/2);
cout<<"mse is "<<fixed<<setprecision(6)<<mse<<endl;
psnr=10*log10(65025/mse);
cout<<"psnr is "<<psnr<<endl;
cout<<"vqm hv gain is "<<(meanHV*0.2483)<<endl;
cout<<"vqm hv loss is "<<(meanHVBelow*0.5969)<<endl;
cout<<"vqm si gain is "<<(meanSI*-2.3416)<<endl;
cout<<"vqm si loss is "<<(siMean5per*-0.2097)<<endl;
cout<<"vqm coher color is "<<(color10per*0.0192)<<endl;
cout<<"vqm contati is "<<(contati10per*0.0431)<<endl;
vqm=(contati10per*0.0431)+(color10per*0.0192)+(meanHV*0.2483)+(meanHVBelow*0.5969)-
(meanSI*2.3416)-
(siMean5per*0.2097);
cout<<"vgm is "<<vqm<<endl;
time(&rawtime2);
cout<<"program ended at "<<ctime(&rawtime2);
cout<<setprecision(0)<<"program lasted "<<difftime(rawtime2,rawtime)<<" seconds"<<endl<<endl;
//close program
return 0;
}
```