ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη και βελτιστοποιήσεις εργαλείου ανάλυσης βίντεο κωδικοποιημένου σε HEVC

Μπέκας Κωνσταντίνος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Υπεύθυνος
Αθανάσιος Λουκόπουλος
Επίκουρος Καθηγητής

Λαμία, 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Ανάπτυξη και βελτιστοποιήσεις εργαλείου ανάλυσης βίντεο κωδικοποιημένου σε HEVC

Μπέκας Κωνσταντίνος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Αθανάσιος Λουκόπουλος
Επίκουρος Καθηγητής

Λαμία, 2020

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί* ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Ανάπτυξη και βελτιστοποιήσεις εργαλείου ανάλυσης βίντεο κωδικοποιημένου σε HEVC

Μπέκας Κωνσταντίνος

Τριμελής Επιτροπή:

Λουκόπουλος Αθανάσιος, Επίκουρος Καθηγητής του τμήματος Πληροφορικής με Εφαρμογές στην Βιοϊατρική του Πανεπιστημίου Θεσσαλίας (επιβλέπων)

Κακαρούντας Αθανάσιος, Επίκουρος Καθηγητής του τμήματος Πληροφορικής με Εφαρμογές στην Βιοϊατρική του Πανεπιστημίου Θεσσαλίας

Αναγνωστόπουλος Ιωάννης, Αναπληρωτής Καθηγητής του τμήματος Πληροφορικής με Εφαρμογές στην Βιοϊατρική του Πανεπιστημίου Θεσσαλίας

Περίληψη

Η ανάγκη για κωδικοποίηση βίντεο είναι απαραίτητη στις μέρες μας, καθώς το βίντεο εμπλέκεται σε όλες σχεδόν τις δραστηριότητές μας. Μερικές από αυτές είναι η εκπαίδευση, η εργασία, η επικοινωνία και η ψυχαγωγία. Δεδομένου ότι όλες αυτές οι δραστηριότητες πραγματοποιούνται μέσω του Διαδικτύου και οι κάμερες προσφέρουν μεγαλύτερη ποιότητα μέρα με τη μέρα, προέκυψε η ανάγκη για πιο αποτελεσματικά πρότυπα κωδικοποίησης βίντεο. Ένα από αυτά είναι το High Efficiency Video Coding (HEVC) που προσφέρει καλύτερη απόδοση κωδικοποίησης σε σύγκριση με τα προηγούμενα πρότυπα χωρίς απώλεια ποιότητας.

Αυτή η εργασία παρουσιάζει την ανάπτυξη και βελτιστοποιήσεις ενός εργαλείου ανάλυσης βίντεο που αποκωδικοποιεί και αναπαράγει ένα βίντεο κωδικοποιημένο στο πρότυπο HEVC. Ο σκοπός της εφαρμογής είναι να παρέχει στον χρήστη οπτικές και στατιστικές πληροφορίες σχετικά με τις δομές κωδικοποίησης που εφαρμόστηκαν από το HEVC μέσω γραφικής διεπαφής χρήστη (GUI).

Abstract

Need for video coding is necessary in our days, since video is involved in almost all of our activities. Some of them are: Education, work, communication, entertainment. Since all of these activities take place over the internet and the cameras offer greater quality day by day, the need of more effective video coding standards came up. One of them is High Efficiency Video Coding (HEVC) which offers better coding efficiency without loosing quality.

This thesis presents the development and optimizations of a video analysis tool that decodes and reproduces a video coded in HEVC standard. The purpose of the application is to provide to user optical and statistical information about the coding structures that implemented by HEVC through graphical user interface (GUI).

Περιεχόμενα

Πίνακας εικόνων

1. Εισαγωγή

Η παρακάτω εργασία βασίστηκε στην μεταπτυχιακή διπλωματική εργασία του κ. Γρηγόριου Δημόπουλου [1].
Ο σκοπός αυτής της εργασίας είναι η ανάπτυξη και οι βελτιστοποιήσεις μιας εφαρμογής σε περιβάλλον εργασίας Microsoft Windows, η οποία όχι μόνο θα αποκωδικοποιεί και θα αναπαράγει βίντεο κωδικοποιημένο σε HEVC, αλλά θα απεικονίζει τη δομή των slices, tiles καθώς και των CTUs και των CUs. Ταυτόχρονα θα απεικονίζει πληροφορίες σχετικές με την ακολουθία βίντεο, πληροφορίες που προκύπτουν κατά την διάρκεια της αποκωδικοποίησης και αφορούν τις παραμέτρους σύμφωνα με τις οποίες κωδικοποιήθηκε το βίντεο, πληροφορίες για τις δομές που περιέχει κάθε στιγμιότυπο του βίντεο καθώς και στατιστικά για τον τρόπο πρόβλεψης που αφορούν τα CUs εντός ενός στιγμιότυπου. Συνοπτικά οι λειτουργίες που προστέθηκαν στην εφαρμογή είναι οι εξής:

- Προσαρμογή του μεγέθους του πλαισίου εμφάνισης βίντεο σύμφωνα με τις προτιμήσεις του χρήστη.
- Οπτικοποίηση της διαίρεσης των πλαισίων βίντεο σε Coding Units (CUs) ανάλογα με τις παραμέτρους διαμέρισης που έχουν οριστεί κατά την διαδικασία κωδικοποίησης.
- Προσθήκη ενός μπλοκ που εμφανίζει πληροφορίες που αφορούν την ακολουθία του βίντεο.
- Προσθήκη ενός μπλοκ που παρουσιάζει πληροφορίες για το τρέχον στιγμιότυπο του βίντεο.
- Προσθήκη ενός μπλοκ που εμφανίζει στατιστικά που αφορούν τα CUs στο τρέχον στιγμιότυπο.
- Προτέθηκε η λειτουργία σύγκρισης ενός κωδικοποιημένου βίντεο με το αρχικό ασυμπίεστο έτσι ώστε να υπολογιστεί το PSNR, ένας δείκτης της ποιότητας κωδικοποίησης του βίντεο.
- Προστέθηκε η λειτουργία επισήμανσης των Intra κωδικοποιημένων CUs με την χρήση γραφικών.
- Προστέθηκε η λειτουργία επισήμανσης των Inter κωδικοποιημένων CUs με την χρήση γραφικών.
- Προστέθηκε η λειτουργία επισήμανσης των CUs που έγιναν merge με την χρήση γραφικών.

Για την ανάπτυξη και τις βελτιστοποιήσεις της εφαρμογής “HEVC Video Analysis Tool” [1] χρησιμοποιήθηκαν τα παρακάτω προγράμματα και εργαλεία:

- HEVC encoder and decoder and the HM reference software version 16.15 [2] , το οποίο αναπτύχθηκε από την σύμπραξη Joint Collaborative Team on Video Coding (JCT-VC) των ITU-T SG16 WP3 και ISO/IEC JTC1/SC29/WG11.

- Το Microsoft Visual Studio 2019 [3] για τον σχεδιασμό και τον προγραμματισμό της κύριας γραφικής διεπαφής χρήστη (GUI) για περιβάλλον εργασίας Microsoft Windows.

- Η OpenCV [4], η οποία είναι μια βιβλιοθήκη ανοιχτού κώδικα υπολογιστικής όρασης, που χρησιμοποιήθηκε για την ανάγνωση των YUV εικόνων.

- C [5] και C++ [6] γλώσσες προγραμματισμού για την ανάπτυξη του προγράμματος.

- Το draw.io [7] για τη δημιουργία των διαγραμμάτων.

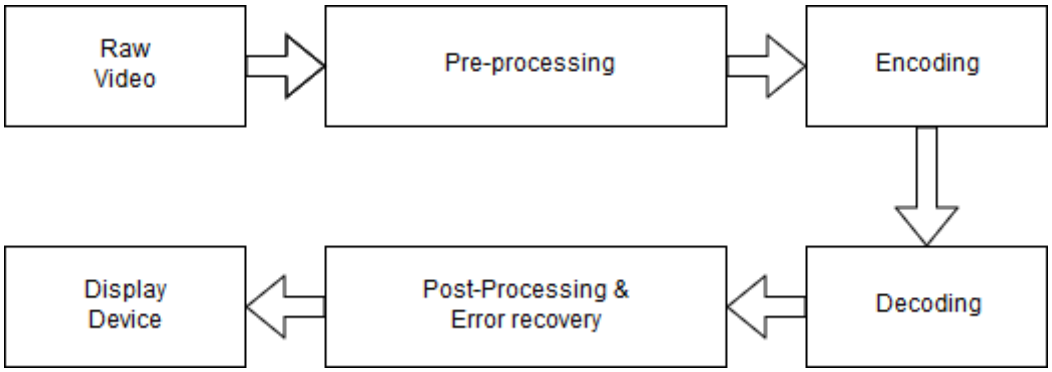
Η εργασία αυτή οργανώνεται ως εξής: Το Κεφάλαιο 2 περιέχει μία σύντομη επισκόπηση της ιστορίας της κωδικοποίησης βίντεο. Επίσης αναλύει τις βασικές ιδέες της διαδικασίας κωδικοποίησης βίντεο. Το Κεφάλαιο 3 κάνει μια εισαγωγή στο HEVC και επισημαίνονται οι διαφορές με τον πρόγονό του, το H.264. Επιπλέον παρουσιάζεται μια επισκόπηση του HM reference software καθώς και του αρχείου παραμετροποίησης του κωδικοποιητή. Το κεφάλαιο 4 παρέχει πληροφορίες σχετικά με τις δομές μπλοκ που χρησιμοποιήθηκαν από το πρότυπο H.265/HEVC. Το κεφάλαιο 5 περιέχει λεπτομερή περιγραφή της ανάπτυξης του προγράμματος καθώς και την παρουσίαση αυτού. Τέλος το κεφάλαιο 6 συνοψίζει και ολοκληρώνει αυτή την εργασία.

2. Επισκόπηση συμπίεσης βίντεο

2.1. Κωδικοποίηση βίντεο

Τις τελευταίες δεκαετίες η χρήση του βίντεο έχει μπει στην καθημερινότητά μας. Η τεχνολογική εξέλιξη έχει δημιουργήσει την ανάγκη για την συμπίεση/κωδικοποίηση του βίντεο, καθώς το μέγεθος που απαιτείται για την αποθήκευσή του ασυμπίεστου video αυξάνεται λογαριθμικά με το πέρας του χρόνου και απαιτεί μεγάλο κόστος και χρόνο για να αποσταλεί.

Για να αντιμετωπιστούν αυτά τα ζητήματα αναπτύχθηκαν κάποιες μορφές κωδικοποίησης/συμπίεσης όπου με βάση μαθηματικούς μετασχηματισμούς αφαιρείται η περιττή πληροφορία στην εικόνα, που δεν είναι αντιληπτή στον άνθρωπο. Για να αναπαραχθεί το βίντεο πρέπει στην συνέχεια να αποκωδικοποιηθεί/αποσυμπιεστεί. Η διαδικασία αυτή της κωδικοποίησης-αποκωδικοποίησης πραγματοποιείται μέσω του Codec, το οποίο αποτελείται από τον encoder και τον decoder αντίστοιχα. Το codec μπορεί να είναι υλοποιημένο σε software είτε σε hardware.



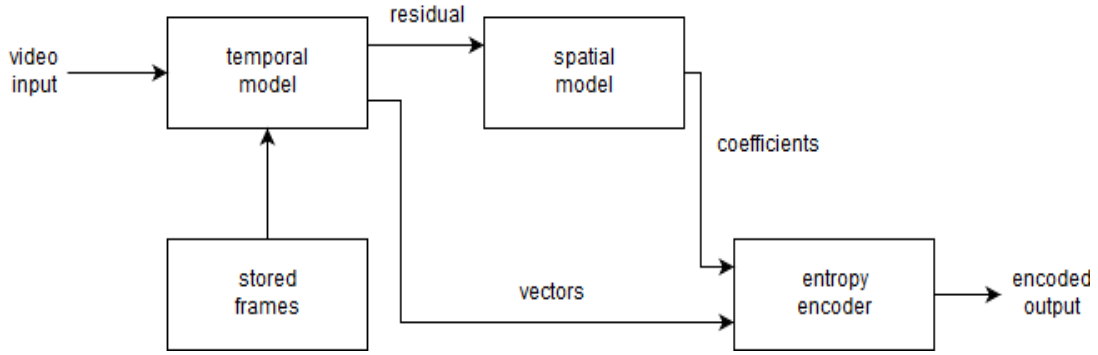
Εικόνα 1:Video Codec – η βασική δομή

Διαφορετικές υλοποιήσεις κωδικοποίησης βίντεο διαφέρουν ως προς τις προδιαγραφές τους και τον τρόπο λειτουργίας τους. Έτσι αυτές μπορούν να περιγραφούν ως video coding formats. Για να μπορέσει ένα format να εδραιωθεί πρέπει να πληροί κάποιες προδιαγραφές. Αυτές τις προδιαγραφές τις ορίζουν διεθνείς οργανισμοί όπως το ITU (International Communication Union) και το ISO (International Organization for Standardization).

2.2. Αρχές κωδικοποίησης βίντεο

Οι αλγόριθμοι συμπίεσης βίντεο λειτουργούν αφαιρώντας τον πλεονασμό πληροφορίας σε χρονικό, χωρικό πεδίο καθώς και στο πεδίο των συχνοτήτων. Αφαιρώντας τον πλεονασμό από διαφορετικά πεδία επιτυγχάνεται συμπίεση σημαντικού βαθμού με συγκεκριμένο κόστος απώλειας πληροφορίας. Για να επιτευχθεί περαιτέρω συμπίεση, κωδικοποιείται η επεξεργασμένη πληροφορία χρησιμοποιώντας έναν κωδικοποιητή εντροπίας. Σημαντικό είναι ότι σε αυτό το στάδιο δεν υπάρχει απώλεια πληροφορίας.

Ένα CODEC κωδικοποιεί μία ακολουθία βίντεο σε συμπιεσμένη μορφή και την αποκωδικοποιεί έτσι ώστε να δημιουργήσει ένα αντίγραφο της αρχικής ακολουθίας. Εάν το αντίγραφο είναι πανομοιότυπο του αρχικού, τότε επιτυγχάνεται κωδικοποίηση χωρίς απώλειες (lossless), αλλιώς εάν η αποκωδικοποιημένη ακολουθία διαφέρει από την αρχική, τότε η διαδικασία έχει απώλεια (lossy).



Εικόνα 2: Η δομή του κωδικοποιητή

Ο κωδικοποιητής αποτελείται από τρεις βασικές μονάδες, το χρονικό μοντέλο, το χωρικό μοντέλο και τον κωδικοποιητή εντροπίας.
Η είσοδος στο χρονικό μοντέλο είναι μια ασυμπίεστη ακολουθία βίντεο. Το χρονικό μοντέλο μειώνει τον πλεονασμό, εκμεταλλευόμενο τις ομοιότητες ανάμεσα σε γειτονικά στιγμιότυπα, κατασκευάζοντας μια πρόβλεψη για το τρέχον στιγμιότυπο. Στις σύγχρονες τεχνικές η πρόβλεψη σχηματίζεται από ένα ή περισσότερα προηγούμενα ή μελλοντικά στιγμιότυπα και βελτιώνεται με την αντιστάθμιση των διαφορών ανάμεσα στα στιγμιότυπα αυτά. Η έξοδος από το χρονικό μοντέλο είναι ένα υπολειπόμενο στιγμιότυπο, το οποίο δημιουργήθηκε αφαιρώντας την πρόβλεψη από το αρχικό, καθώς και ένα σύνολο διανυσμάτων κίνησης τα οποία περιγράφουν πώς η κίνηση αντισταθμίστηκε.

Το υπολειπόμενο στιγμιότυπο στη συνέχεια οδηγείται στο χωρικό μοντέλο, το οποίο χρησιμοποιεί τις ομοιότητες μεταξύ γειτονικών περιοχών στο τρέχον στιγμιότυπο, για να μειώσει τον χωρικό πλεονασμό. Αυτό επιτυγχάνεται με την εφαρμογή ενός μετασχηματισμού στα στιγμιότυπα αυτά, τα οποία στη συνέχεια κβαντίζονται. Ο μετασχηματισμός αυτός μεταφέρει την εικόνα σε άλλο πεδίο όπου πλέον αναπαρίσταται από τους συντελεστές μετασχηματισμού. Οι συντελεστές αυτοί

κβαντίζονται έτσι ώστε να αφαιρεθεί η περιττή πληροφορία, η οποία δεν είναι αντιληπτή από το HVS (Human Visual System) [8]. Η έξοδος του χωρικού μοντέλου είναι ένα σύνολο από κβαντισμένους συντελεστές μετασχηματισμού.

Οι παράμετροι του χρονικού μοντέλου (διανύσματα κίνησης) καθώς και του χωρικού μοντέλου επίσης(συντελεστές μετασχηματισμού) συμπιέζονται από τον κωδικοποιητή εντροπίας, ο οποίος αφαιρεί τον στατιστικό πλεονασμό στα δεδομένα και δημιουργεί μια συμπιεσμένη ροή δεδομένων. Μία συμπιεσμένη ακολουθία αποτελείται από κωδικοποιημένα διανύσματα κίνησης, κωδικοποιημένους συντελεστές μετασχηματισμού και πληροφορίες κεφαλίδας.

Ο αποκωδικοποιητής ανακατασκευάζει το στιγμιότυπο από την συμπιεσμένη ροή δεδομένων. Οι παράμετροι και τα διανύσματα κίνησης αποκωδικοποιούνται από τον αποκωδικοποιητή εντροπίας, όπου στη συνέχεια το χωρικό μοντέλο αποκωδικοποιείται ώστε να ανακατασκευαστεί το υπολειπόμενο στιγμιότυπο. Ο αποκωδικοποιητής χρησιμοποιεί τα διανύσματα κίνησης μαζί με ένα ή περισσότερα προηγούμενα στιγμιότυπα για να δημιουργήσει την πρόβλεψη του τρέχοντος στιγμιότυπου. Τέλος αυτό ανακατασκευάζεται προσθέτοντας το υπολειπόμενο με την πρόβλεψη.

Κατά τον σχεδιασμό ενός Codec πρέπει να λαμβάνονται υπόψιν οι παρακάτω παράμετροι [8]:

- **Ποιότητα**
  Η ποιότητα του συμπιεσμένου βίντεο θα πρέπει να διατηρείται σύμφωνα με τις απαιτήσεις της κάθε εφαρμογής. Ένα εργαλείο που μπορεί να χρησιμοποιηθεί για τον υπολογισμό της ποιότητας του συμπιεσμένου βίντεο είναι το PSNR (Peak Signal-to-Noise Ratio), το οποίο μετράει το μέγιστο σφάλμα ανάμεσα στην κωδικοποιημένη εικόνα και την αρχική.

  $$PSNR(dB) = 20\log_{10}\frac{MAX}{\sqrt{MSE}}$$

  Η τιμή MAX αναπαριστά την μέγιστη τιμή που λαμβάνει το pixel μιας εικόνας. Για εικόνες γκρι κλίμακας χρωματικού βάθους 8 bit η μέγιστη τιμή είναι το 255. Η τιμή MSE (Mean Squared Error), δηλαδή το μέσο τετραγωνικό σφάλμα είναι το αθροιστικό σφάλμα μεταξύ της συμπιεσμένης και της ανακατασκευασμένης εικόνας.

  $$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}[I(i,j) - I'(i,j)]^2$$

Όπου MxN είναι οι διαστάσεις τις εικόνας, το I συμβολίζει την αρχική ασυμπίεστη εικόνα και το I’ την συμπιεσμένη εικόνα.

- **Χρονική πολυπλοκότητα**
  Η πολυπλοκότητα του αλγορίθμου κωδικοποίησης είναι ένας σημαντικός παράγοντας που θα πρέπει να ληφθεί υπόψιν καθώς η ταχύτητα κωδικοποίησης δε θα πρέπει να είναι χαμηλή.

- **Λόγος συμπίεσης**
  Ο λόγος συμπίεσης ορίζεται ως ο λόγος των bits που χρειάζονται να αναπαραστήσουν το βίντεο πριν τη συμπίεση προς τον αριθμό των bits που χρειάζονται μετά τη συμπίεση.

$$Compression\ Ratio = \frac{μέγεθος\ του\ αρχικού\ βίντεο}{μέγεθος\ του\ συμπιεσμένου\ βίντεο}$$

- **Ρυθμός μετάδοσης**
  Ο ρυθμός μετάδοσης είναι άλλος ένας τρόπος αξιολόγησης των τεχνικών συμπίεσης. Μονάδα μέτρησής του είναι το bit/s.

$$Bitrate = \frac{Μέγεθος\ του\ βίντεο}{Διάρκεια\ του\ βίντεο}$$

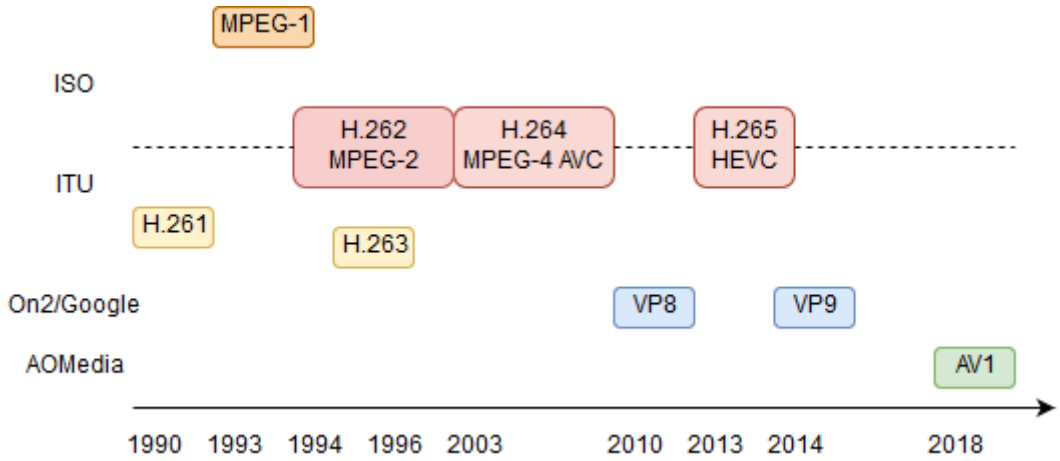
2.3. Πρότυπα κωδικοποίησης βίντεο

Πολλές από τις βασικές έννοιες της κωδικοποίησης βίντεο, όπως η κωδικοποίηση μετασχηματισμού, η εκτίμηση κίνησης, η αντιστάθμιση καθώς και η κωδικοποίηση εντροπίας αναπτύχθηκαν στην δεκαετία του 1970 και του 1980.

Η εικόνα 3 δείχνει τα πιο σημαντικά πρότυπα κωδικοποίησης που έχουν αναπτυχθεί μέχρι σήμερα. Οι δύο βασικοί οργανισμοί που συνέβαλλαν στην ανάπτυξη προτύπων κωδικοποίησης βίντεο είναι ο Διεθνής Οργανισμός Τυποποίησης (ISO) και ο Τομέας Τυποποίησης Τηλεπικοινωνιών της Διεθνούς Ένωσης Τηλεπικοινωνιών (ITU-T). Ο Διεθνής Οργανισμός Τυποποίησης (ISO) απαρτίζεται από εκπροσώπους διαφόρων εθνικών οργανισμών τυποποίησης και σκοπός του είναι η προώθηση παγκόσμιων βιομηχανικών και εμπορικών προτύπων. Υποομάδα του ISO αποτελεί η Motion Picture Experts Group (MPEG), η οποία σχεδίασε και ανέπτυξε πρότυπα όπως τα MPEG-1, MPEG-2 και MPEG-4. Ο τομέας τυποποίησης τηλεπικοινωνιών της

Διεθνούς Ένωσης Τηλεπικοινωνιών (ITU-T) έχει στόχο την ανάπτυξη προτύπων που χρησιμοποιούνται στις τηλεπικοινωνίες. Τα πρότυπα που έχει αναπτύξει η (ITU-T) είναι τα .26x που σχεδιάστηκαν για να χρησιμοποιηθούν στην βιντεοτηλεφωνία.

Το πρώτο πρακτικό πρότυπο ήταν το H.261 της ITU-T. Η Motion Picture Experts Group αργότερα κυκλοφόρησε το MPEG-1 εν έτη 1991 και ακολούθησε το MPEG-2 καθώς και το H.262 το 1994.
Μετά από ένα διάστημα 9 ετών κυκλοφόρησε το H.264/MPEG-4 AVC, το οποίο χρησιμοποιείται ευρέως μέχρι και σήμερα από μεγάλους τεχνολογικούς οργανισμούς που απευθύνονται στην πλειοψηφία των χρηστών. Τέλος, η τεχνολογική εξέλιξη και η ανάγκη για κωδικοποίηση βίντεο μεγάλης ευκρίνειας οδήγησε στη δημιουργία του H.265/MPEG-H HEVC το 2013.



Εικόνα 3: Τα πρότυπα κωδικοποίησης ανά τον χρόνο

Η πλειοψηφία των προτύπων κωδικοποίησης βίντεο βασίζεται στην υβριδική τεχνική κωδικοποίησης βίντεο με χρήση μπλοκ, η οποία εφαρμόστηκε πρώτη φορά στο H.161 (εικόνα 4) και υιοθετήθηκε από όλα τα επόμενα πρότυπα. Παρά το γεγονός ότι όλα τα πρότυπα έχουν κοινή βασική δομή, η απόδοση των προτύπων αυτών παρουσιάζει σημαντική βελτίωση από γενιά σε γενιά. Παρακάτω παρουσιάζονται συνοπτικά τα καθιερωμένα πρότυπα κωδικοποίησης:

- **H.261**
  Το H.261 ορίστηκε από την ITU το 1990 και ήταν το πρώτο ευρέως αποδεκτό πρότυπο κωδικοποίησης. Χρησιμοποιήθηκε κυρίως για τηλεδιασκέψεις και βιντεοτηλεφωνία μέσω ISDN. Υποστηρίζει δύο μορφές εικόνας: την CIF (Common Intermediate Format) και την QCIF (Quarter of Common Intermediate Format). Τα βήματα που ακολουθούνται για την κωδικοποίηση είναι η πρόβλεψη, ο μετασχηματισμός μπλοκ, η κβάντιση και η κωδικοποίηση εντροπίας. Το H.261 χρησιμοποιεί έναν αλγόριθμο αντιστάθμισης κίνησης που κωδικοποιεί τις διαφορές μεταξύ γειτονικών στιγμιότυπων έτσι ώστε να μειώσει τον χρονικό πλεονασμό. Σχεδιάστηκε για εφαρμογές

βιντεοτηλεφωνίας, κάτι που απαιτεί χαμηλή καθυστέρηση και σταθερό ρυθμό μετάδοσης.

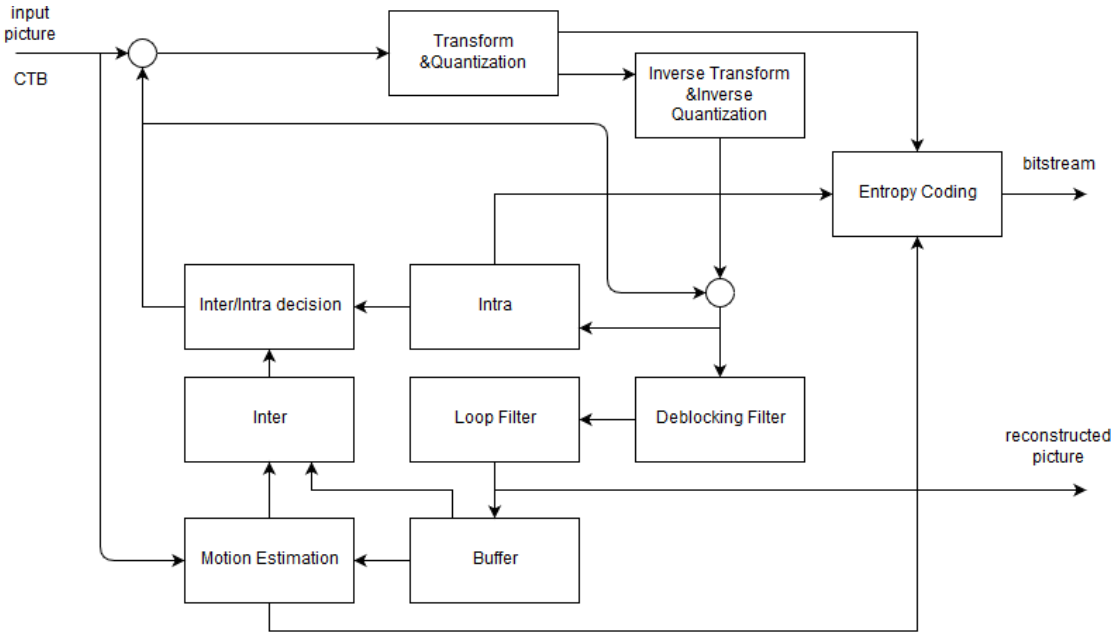
- **MPEG-1**
  Το MPEG-1 [8]που κυκλοφόρησε το 1993 είναι το πρώτο πρότυπο συμπίεσης βίντεο του ISO. Έχει πολλά κοινά με το H.261 αλλά με κάποιες βελτιώσεις. Υποστηρίζει κωδικοποίηση με ρυθμό περίπου 1.5Mbps. Αυτό το πρότυπο σχεδιάστηκε για αποθήκευση και ανάκτηση αρχείων ήχου και βίντεο σε ψηφιακά μέσα αποθήκευσης. Το κύριο πλεονέκτημά του απέναντι στον πρόγονό του, το H.261 είναι ότι χρησιμοποιεί μέθοδο πρόβλεψης διπλής κατεύθυνσης με αποτέλεσμα την μείωση του θορύβου.

- **H.262/MPEG-2**
  Το H.262/MPEG-2 [9] [8]κυκλοφόρησε το 1994 και καθορίστηκε από κοινού από τον ISO και την ITU. Το MPEG-2 βασίστηκε πάνω στο MPEG-1 με κάποιες βελτιώσεις, όπως η ευρύτερη αντιστάθμιση κίνησης καθώς και υποστήριξη αλληλοσυνδεόμενων βίντεο. Το MPEG-2 υποστηρίζει ρυθμό μετάδοσης 9.8Mbps και αναλύσεις έως 720x480pixels.

- **H.263**
  Το H.263 [8] είναι πρότυπο της ITU και αναπτύχθηκε το 1995 ως βελτίωση του H.261 προσθέτοντας χαρακτηριστικά όπως: εκτίμηση κίνησης μισού pixel, εξειδικευμένη λειτουργία πρόβλεψης και μη περιοριζόμενα διανύσματα κίνησης. Το H.263 υποστηρίζει πέντε μορφές εικόνας: Την Sub-QCIF (128x96), QCIF (176x144), CIF (352x288), 4-CIF (704x576) και την 16-CIF (1408x1152). Το H.263 εστίασε στην βιντεοτηλεφωνία μέσω PSTN. Η αποδοτικότητα του αλγορίθμου είναι καλύτερη από αυτήν του H.261.

- **H.264/MPEG-4-AVC**
  Το H.264/MPEG-4 [8] [10] AVC κυκλοφόρησε το 2003 από την συνεργασία της Video Coding Experts Group (VCEG) , ομάδας της ITU-T και της Moving Picture Experts Group (MPEG) , ομάδας της ISO/IEC. Το H.264/MPEG-4 δείχνει σημαντική βελτίωση στην αποδοτικότητα της inter/intra κωδικοποίησης. Εμφανίζει ενισχυμένη αντοχή σφάλματος και μεγαλύτερη ευελιξία. Επίσης έχει αποδοτική αντιστάθμιση καθώς και μειωμένο ρυθμό μετάδοσης. Τα μπλοκ που χρησιμοποιούνται για την αντιστάθμιση κίνησης έχουν διαφορετικά μεγέθη, κάτι που σημαίνει καλύτερη ποιότητα βίντεο. Η βασική μονάδα επεξεργασίας είναι μακρομπλόκ μεγέθους 16x16 pixels. Εφαρμογές του προτύπου αυτού είναι εκπομπές τηλεόρασης υψηλής ευκρίνειας (HDTV) , μετάδοση βίντεο μέσω internet και βιντεοσυσκέψεις.

- **H.265/HEVC**
  Το H.265/HEVC [11] [12] [9] (High Efficiency Video Coding) αναπτύχθηκε από την Joint Collaborative Team on Video Coding (JCT-VC), ομάδα που δημιουργήθηκε από την συνεργασία των ITU και ISO/IEC το 2013. Μέχρι σήμερα είναι το καλύτερο πρότυπο, επειδή παρέχει αρκετά νέα χαρακτηριστικά σε σύγκριση με προηγούμενα πρότυπα. Κάποια από αυτά είναι μεγαλύτερες δομές μπλοκ κωδικοποίησης (64x64 pixels), δομές μπλοκ σε μορφή τετραδικού δέντρου, εξειδικευμένη πρόβλεψη διανυσμάτων κίνησης και περισσότερες λειτουργίες κατευθύνσεων στην intra πρόβλεψη. Περισσότερες λεπτομέρειες για το HEVC θα παρουσιαστούν στο επόμενο κεφάλαιο.



Εικόνα 4: Η βασική δομή του αποκωδικοποιητή από το H.261 και μετά

3. High Efficiency Video Coding (HEVC)

Το HEVC (High Efficiency Video Coding) είναι ο διάδοχος του H.264/MPEG-4 AVC και αναπτύχθηκε από κοινού από την ISO/IEC Moving Picture Experts Group (MPEG) και από την ITU-T Video Coding Experts Group (VCEG). Το 2010 αυτές οι δύο ομάδες επισήμως ξεκίνησαν την ανάπτυξη του H.265 HEVC υπό την ονομασία JCT-VC (Joint Collaborative Team on Video Coding) [8]. Η πρώτη έκδοση του HEVC ολοκληρώθηκε τρία χρόνια αργότερα, τον Ιανουάριο του 2013 και ορίστηκε επίσημα τον Απρίλιο του 2013. Σήμερα έχει τυποποιηθεί ως ITU-T Recommendation H.265 και ISO/IEC International Standard 23008-2 (MPEG-H part 2).

Το HEVC σχεδιάστηκε έτσι ώστε να καλύπτει όλες τις υπάρχουσες εφαρμογές του H.264 / MPEG-4 AVC και να εστιάσει ιδιαίτερα σε τρία βασικά ζητήματα, την σημαντική βελτίωση της απόδοσης κωδικοποίησης, την αυξημένη ανάλυση βίντεο που προσφέρουν οι κάμερες πλέον καθώς και τη χρήση αρχιτεκτονικών παράλληλης επεξεργασίας [11]. Όπως συμβαίνει και με όλα τα προηγούμενα πρότυπα κωδικοποίησης βίντεο των ITU-T και ISO/IEC έτσι και στο HEVC η δομή του bitstream και το συντακτικό είναι τυποποιημένα καθώς και οι περιορισμοί στο bitstream και η χαρτογράφησή του για την παραγωγή αποκωδικοποιημένων εικόνων. Για να βοηθηθεί η κοινότητα του κλάδου να μάθει πώς να χρησιμοποιεί το πρότυπο αυτό δεν παρέχεται μόνο το έγγραφο προδιαγραφών του προτύπου αλλά και ο πηγαίος κώδικας του HM reference software ως παράδειγμα του τρόπου κωδικοποίησης και αποκωδικοποίησης του βίντεο HEVC.

Το HEVC έχει τον διπλάσιο συντελεστή συμπίεσης σε σχέση με το H.264/MPEG-4 AVC χωρίς να αλλάζει η ποιότητα του βίντεο. Μπορεί εναλλακτικά να χρησιμοποιηθεί για την παροχή βελτιωμένης ποιότητας βίντεο στο ίδιο bitrate, επίσης μπορεί να υποστηρίξει 8K βίντεο και αναλύσεις μέχρι 8192x4320 pixel.

3.1. Τα κύρια χαρακτηριστικά του HEVC

Όπως σε όλα τα προηγούμενα πρότυπα κωδικοποίησης βίντεο των ITU-T και ISO/IEC JTC 1 από το H.261, ο σχεδιασμός του HEVC ακολουθεί την προσέγγιση της υβριδικής κωδικοποίησης βίντεο βασιζόμενη σε μπλοκ**.** Ο βασικός αλγόριθμος κωδικοποίησης είναι ένα υβρίδιο inter πρόβλεψης έτσι ώστε να εκμεταλλευτεί τον χρονικό στατιστικό πλεονασμό καθώς και intra πρόβλεψης για να εκμεταλλευτεί τον χωρικό στατιστικό πλεονασμό. Σημαντικό είναι ότι δεν υπάρχει κάποιο ενιαίο στοιχείο στο HEVC όπου να οφείλεται σε αυτό το μεγαλύτερο μέρος της σημαντικής βελτίωσης της αποτελεσματικότητας της συμπίεσης σε σχέση με τα προηγούμενα πρότυπα. Είναι μια πληθώρα πολλών επιμέρους βελτιώσεων και προσθηκών που συμβάλλουν σε αυτό το σημαντικό κέρδος.

3.1.1. Video Coding Layer

Η τυπική διαδικασία κωδικοποίησης που ακολουθεί ένας αλγόριθμος παραγωγής bitstream, το οποίο είναι συμβατό με τις προδιαγραφές του HEVC έχει ως εξής.
Κάθε κωδικοποιημένο στιγμιότυπο ή εικόνα διασπάται σε μπλοκ με τον ακριβή διαχωρισμό μπλοκ να μεταφέρεται στον αποκωδικοποιητή. Το πρώτο στιγμιότυπο ενός βίντεο κωδικοποιείται μόνο χρησιμοποιώντας Intra πρόβλεψη, δηλαδή πρόβλεψη δεδομένων χωρικά από περιοχή σε περιοχή της εικόνας. Στα στιγμιότυπα που ακολουθούν χρησιμοποιείται κατά κανόνα Inter πρόβλεψη. Η διαδικασία κωδικοποίησης για την Inter πρόβλεψη αποτελείται από την επιλογή δεδομένων κίνησης που περιλαμβάνουν το επιλεγμένο στιγμιότυπο καθώς και το διάνυσμα κίνησης (Motion Vector) που πρόκειται να εφαρμοστεί για την πρόβλεψη δειγμάτων κάθε μπλοκ. Ο κωδικοποιητής και ο αποκωδικοποιητής παράγουν πανομοιότυπα σήματα inter πρόβλεψης εφαρμόζοντας αντιστάθμιση κίνησης χρησιμοποιώντας τα διανύσματα κίνησης και τα δεδομένα απόφασης τρόπου πρόβλεψης, τα οποία μεταδίδονται στον αποκωδικοποιητή.
Το υπολειπόμενο σήμα της Intra/Inter πρόβλεψης σχηματίζεται αφαιρώντας το αρχικό μπλοκ από την πρόβλεψη αυτού. Το υπολειπόμενο σήμα μετασχηματίζεται από ένα γραμμικό χωρικό μετασχηματισμό από όπου προκύπτουν οι συντελεστές μετασχηματισμού. Οι συντελεστές αυτοί στην συνέχεια κλιμακώνονται , κβαντίζονται ,κωδικοποιούνται και μεταδίδονται μαζί με τις πληροφορίες πρόβλεψης.

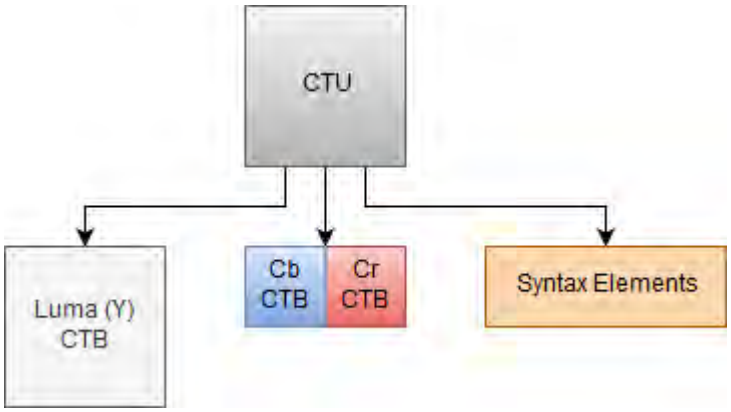
Ο κωδικοποιητής εκτελεί επίσης την λειτουργία του αποκωδικοποιητή έτσι ώστε να παραχθούν πανομοιότυπες προβλέψεις για τα επόμενα δεδομένα. Επομένως οι κβαντισμένοι συντελεστές μετασχηματισμού ανακατασκευάζονται χρησιμοποιώντας αντίστροφη κλιμάκωση και τότε γίνεται αντίστροφος μετασχηματισμός έτσι ώστε να αντιγράψουν την αποκωδικοποιημένη προσέγγιση του υπολειπόμενου σήματος. Το υπολειπόμενο τότε προστίθεται στην πρόβλεψη και στην συνέχεια τροφοδοτείται σε ένα ή δύο φίλτρα βρόγχου (Loop Filers) για να εξομαλυνθούν τα αντικείμενα που προκύπτουν από την επεξεργασία και τον κβαντισμό των μπλοκ κατά συστάδες. Η τελική απεικόνιση του στιγμιότυπου, το οποίο είναι αντίγραφο της εξόδου του αποκωδικοποιητή αποθηκεύεται σε μία προσωρινή μνήμη για να χρησιμοποιηθεί για την πρόβλεψη των επόμενων.

Τα βίντεο ως προς κωδικοποίηση HEVC γενικά αναμένεται να είναι είσοδος ως στιγμιότυπα προοδευτικής σάρωσης. Δεν υπάρχουν σαφή χαρακτηριστικά κωδικοποίησης στον σχεδιασμό του HEVC που να υποστηρίζουν πεπλεγμένη σάρωση καθώς δεν χρησιμοποιείται πλέον από τις συσκευές αναπαράστασης. Ωστόσο παρέχεται ένα συντακτικό μεταδεδομένων στο HEVC που επιτρέπει σε έναν κωδικοποιητή να δηλώσει ότι έχει στείλει ένα βίντεο πεπλεγμένα σαρωμένο κωδικοποιώντας κάθε πεδίο του πεπλεγμένου βίντεο ως ξεχωριστή εικόνα είτε κωδικοποιώντας κάθε πεπλεγμένα σαρωμένο στιγμιότυπο ως κωδικοποιημένο στην μορφή HEVC. Αυτό συνιστά μία αποτελεσματική μέθοδο χειρισμού πεπλεγμένα

σαρωμένου βίντεο χωρίς να επιβαρύνονται οι αποκωδικοποιητές και χωρίς την ανάγκη για ειδικές διαδικασίες αποκωδικοποίησης για αυτό.

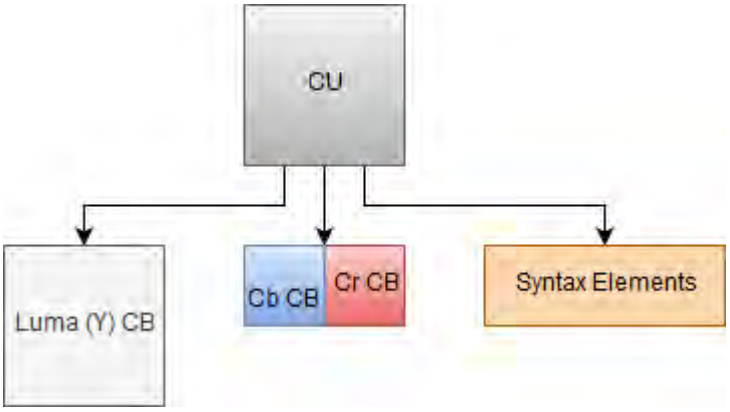
Παρακάτω, τα σημαντικότερα χαρακτηριστικά του εμπλέκονται στην υβριδική κωδικοποίηση βίντεο HEVC [11] [12] [9].

1. **Coding Tree Units (CTU) και Coding Tree Block (CTB):** Στα προηγούμενα πρότυπα κωδικοποίησης η βασική μονάδα κωδικοποίησης ήταν το macroblock, το οποίο στην πιο διαδεδομένη μορφή εικόνας, την 4:2:0 αποτελείται από ένα luma (Y) μπλοκ διαστάσεων 16x16 που φέρει πληροφορίες για την φωτεινότητα και δύο μπλοκ (Cb & Cr) διαστάσεων 8x8 που φέρουν χρωματικά δείγματα. Στο HEVC η ανάλογη δομή είναι το Coding Tree Unit (CTU), του οποίου το μέγεθος επιλέγεται από τον κωδικοποιητή και μπορεί να είναι μεγαλύτερο από το παραδοσιακό macroblock. Το CTU αποτελείται από το luma CTB καθώς και δύο ανάλογα chroma CTBs και τα συντακτικά στοιχεία που το συνοδεύουν (εικόνα 5). Το μέγεθος L x L ενός luma CTB μπορεί να πάρει τις τιμές L=16,32 ή 64 με τα μεγαλύτερα μεγέθη συνήθως να επιτυγχάνεται καλύτερη συμπίεση. Το HEVC υποστηρίζει επίσης την διαμέριση των CTUs και CTBs σε μικρότερα μπλοκ χρησιμοποιώντας την δομή τετραδικού δέντρου.

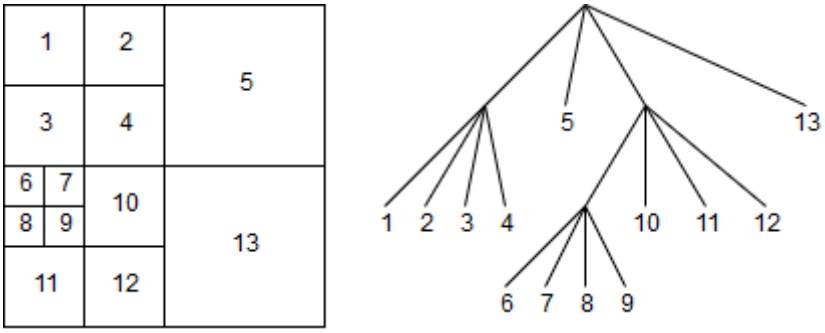


Εικόνα 5: Η δομή του CTU

2. **Coding Units (CUs) και Coding Blocks (CBs):** Η δομή τετραδικού δέντρου του CTU (εικόνα 7) καθορίζει τα μεγέθη και τις τοποθεσίες των luma και chroma CBs. Η ρίζα του τετραδικού δέντρου συσχετίζεται με το CTU, έτσι το μέγεθος του luma CTB είναι το μεγαλύτερο υποστηριζόμενο μέγεθος για ένα luma CB. Η διάσπαση ενός CTU σε luma και chroma CBs σηματοδοτείται από κοινού. Ένα luma CB και δύο chroma CBs μαζί με τα συντακτικά στοιχεία σχηματίζουν ένα CU (εικόνα 6). Ένα CTB μπορεί να περιέχει μόνο ένα CU ή μπορεί να διασπαστεί για να σχηματίσει πολλαπλά CUs και κάθε CU έχει μια συσχετισμένη κατανομή σε μονάδες πρόβλεψης (PUs) και ένα δέντρο μονάδων μετασχηματισμού (TUs).



Εικόνα 6: Η δομή του CU



Εικόνα 7: Παράδειγμα δομής τετραδικού δέντρου

3. **Prediction Units (PUs) και Prediction Blocks (PBs):** Η απόφαση για το εάν μία περιοχή εικόνας κωδικοποιηθεί χρησιμοποιώντας Inter ή Intra πρόβλεψη λαμβάνεται στο επίπεδο των CUs. Μία δομή κατάτμησης PU έχει την ρίζα της στο επίπεδο του CU. Εξαρτώμενα από την επιλογή του τύπου πρόβλεψης τα luma και chroma CBs μπορούν να διασπαστούν περαιτέρω και να προβλεφθούν από τα luma και chroma μπλοκ πρόβλεψης (PBs). Το HEVC υποστηρίζει μεταβλητά μεγέθη για τα PBs, από 64x64 έως 4x4.

4. **Transform Units (TUs) και Transform Blocks (TBs):** Μία δομή δέντρου μονάδας μετασχηματισμού (TU) έχει την ρίζα της στο επίπεδο του CU. Το υπολειπόμενο luma CB μπορεί να είναι πανομοιότυπο του luma μπλοκ μετασχηματισμού (TB) ή μπορεί να διασπαστεί σε περαιτέρω μικρότερα luma TBs. Το ίδιο ισχύει και για τα chroma TBs. Ένας διακριτός μετασχηματισμός συνημιτόνου ορίζεται για τα τετράγωνα TBs με μεγέθη 4x4, 8x8, 16x16, και 32x32.

5. **Motion Vectors (MVs):** Στο HEVC χρησιμοποιείται εξειδικευμένη πρόβλεψη διανυσμάτων κίνησης (Advanced Motion Vecor Prediction), η οποία βασίζεται στην πληροφορία μεταξύ γειτονικών PBs και του στιγμιότυπου αναφοράς. Μπορεί επίσης να χρησιμοποιηθεί ένας τρόπος συγχώνευσης για κωδικοποίηση (Merge Mode) διανυσμάτων κίνησης (MV),

επιτρέποντας την κληρονομικότητα των MVs από χρονικά ή χωρικά γειτονικά PBs.

6. **Motion compensation:** Στην αντιστάθμιση κίνησης χρησιμοποιείται quarter-sample ακρίβεια για τα διανύσματα κίνησης και 7-tap ή 8-tap φίλτρα για την παρεμβολή θέσεων του κλασματικού δείγματος. Παρόμοια με το H.264/MPEG-4 AVC, χρησιμοποιούνται πολλές εικόνες αναφοράς. Για κάθε PB μπορούν να μεταδοθούν είτε ένα είτε δύο διανύσματα κίνησης, με αποτέλεσμα είτε απλή, είτε διπλή κωδικοποίηση αντίστοιχα.

7. **Intrapicture πρόβλεψη:** Τα αποκωδικοποιημένα δείγματα των γειτονικών μπλοκ χρησιμοποιούνται ως δεδομένα αναφοράς για την χωρική πρόβλεψη σε περιοχές που δεν λαμβάνει μέρος η πρόβλεψη μεταξύ γειτονικών στιγμιοτύπων (inter). Οι τρόποι της intra πρόβλεψης είναι είτε η γωνιακή, είτε η επίπεδη, είτε η DC λειτουργία. Η intra πρόβλεψη υποστηρίζει 35 τρόπους πρόβλεψης σε αντίθεση με τους οκτώ που υποστήριζε η αντίστοιχη σύμφωνα με το προγενέστερο πρότυπο, το H.264/MPEG-4 AVC. Όλες οι μέθοδοι πρόβλεψης χρησιμοποιούν δείγματα αναφοράς από γειτονικά ανακατασκευασμένα μπλοκ.

8. **Κβαντιση:** Όπως και στην περίπτωση του H.264/MPEG-4 AVC, στο HEVC χρησιμοποιείται ομοιόμορφη κβάντιση ανασύνθεσης (Unique form Reconstruction Quantization), με πίνακες κλιμάκωσης κβαντισμού που υποστηρίζονται για τα διάφορα μεγέθη των μπλοκ μετασχηματισμού.

9. **Κωδικοποίηση εντροπίας:** Για την κωδικοποίηση εντροπίας χρησιμοποιείται το Context-based Adaptive Binary Arithmetic Coding (CABAC). Αυτό είναι παρόμοιο με το σχήμα CABAC στο H.264/MPEG-4 AVC, αλλά έχει υποστεί αρκετές βελτιώσεις έτσι ώστε να έχει καλύτερη ταχύτητα παραγωγής καθώς και καλύτερη απόδοση συμπίεσης και να μειώσει τις απαιτήσεις μνήμης. Η κωδικοποίηση εντροπίας είναι μία μέθοδος συμπίεσης χωρίς απώλεια πληροφορίας (lossless).

10. **In-loop deblocking filter:** Είναι παρόμοιο με το φίλτρο που χρησιμοποιήθηκε στο H.264/MPEG-4 AVC τροποποιημένο έτσι ώστε να αποδίδει καλύτερα με τις τεχνικές παράλληλης επεξεργασίας.

11. **Δειγματικά προσαρμοστική μετατόπιση (Sample Adaptive Offset):** Μία μη γραμμική αντιστοίχιση εφαρμόζεται μετά το deblocking φίλτρο μέσα στον βρόχο της inter πρόβλεψης. Σκοπός του είναι να βελτιώσει τις τιμές του ανακατασκευασμένου σήματος χρησιμοποιώντας έναν πίνακα αναζήτησης ο οποίος περιγράφεται από επιπλέον παραμέτρους, αυτές μπορούν να προσδιοριστούν από την ανάλυση του ιστογράμματος στην πλευρά του κωδικοποιητή.

3.1.2. High-Level Syntax Architecture

Οι δομές σύνταξης υψηλού επιπέδου που χρησιμοποιήθηκαν στο H.264/MPEG-4 AVC έτσι ώστε να βελτιώσουν την ανθεκτικότητα στην απώλεια δεδομένων καθώς και την ευελιξία για την λειτουργία σε διάφορες εφαρμογές διατηρήθηκαν και στο HEVC. Μερικές από αυτές είναι οι παρακάτω [11]:

I. **Δομή σετ παραμέτρων:** περιέχει πληροφορίες καθώς και έναν μηχανισμό για την αποστολή αυτών, που είναι ουσιαστικής σημασίας για την διαδικασία της αποκωδικοποίησης. Η έννοια της δομής σετ δεδομένων από το H.264/MPEG-4 AVC ενισχύεται από μία νέα, την δομή παραμέτρων βίντεο (VPS).

II. **Δομή σύνταξης μονάδας NAL:** κάθε δομή σύνταξης τοποθετείται σε ένα πακέτο δεδομένων το οποίο ονομάζεται network abstraction layer. Χρησιμοποιώντας το περιεχόμενο μίας επικεφαλίδας μίας μονάδας NAL είναι εύκολο να εντοπιστεί ο σκοπός των σχετικών δεδομένων.

III. **Slices:** Το slice είναι μία δομή η οποία μπορεί να αποκωδικοποιηθεί ανεξάρτητα από άλλα slice της ίδιας εικόνας. Μπορεί να είναι ολόκληρη η εικόνα είτε περιοχή αυτής. Ένας από τους κύριους σκοπούς του slice είναι ο επανασυγχρονισμός σε περίπτωση απώλειας δεδομένων.

IV. **Συμπληρωματικές πληροφορίες βελτιστοποίησης (SEI) και μεταδεδομένα που παρέχουν πληροφορίες σχετικά με την χρηστικότητα του βίντεο (VUI):** Τα δεδομένα αυτά παρέχουν πληροφορίες σχετικά με τον χρονισμό των εικόνων, την ορθή ερμηνεία του χρωματικού χώρου που χρησιμοποιείται για την αναπαράσταση βίντεο κ.λπ..

3.2. Το HEVC σε σύγκριση με το H.264/MPEG-4 AVC

Το HEVC βασίστηκε στην ιδέα της υβριδικής κωδικοποίησης μπλοκ, δηλαδή στην λογική σύμφωνα με την οποία λειτουργεί και ο πρόγονός του, το H.264 με πολλές επιμέρους βελτιώσεις και παραλλαγές. Στόχος της σχεδίασης του HEVC ήταν η υποστήριξη υψηλότερων αναλύσεων, καθώς και η βελτιστοποίηση του χρόνου κωδικοποίησης και αποκωδικοποίησης. Οι κύριες διαφορές που εντοπίζονται ανάμεσα στο HEVC και το H.264/MPEG-4 [8] [9] [11] [13] AVC είναι οι παρακάτω.

Το H.264/MPEG-4 AVC πέτυχε 50% μείωση του bitrate σε σχέση με τον πρόγονό του, το H.263. Αντίστοιχα το HEVC πέτυχε 40-50% μείωση του bitrate σε σύγκριση με το H.264/MPEG-4 χωρίς να υπάρχει απώλεια ποιότητας στην εικόνα. Το HEVC έχει ως βασική δομή επεξεργασίας το CTU σε αντίθεση με τα macroblock που χρησιμοποιήθηκαν από το H.264. Το macroblock έχει μέγεθος 16x16, ενώ το CTU

μπορεί να λάβει διαστάσεις από 16x16 έως και 64x64, πράγμα που κάνει το HEVC πιο ευέλικτο στην λήψη αποφάσεων για τον τρόπο κωδικοποίησης που πρόκειται να εφαρμοστεί σε μικρές ή μεγάλες περιοχές. Για την διαδικασία εκτίμησης κίνησης το macroblock διασπάται σε υπο-μπλοκ με τις ελάχιστες δυνατές διαστάσεις να είναι 4x4, ενώ το CTU μπορεί να διασπαστεί σε CU, PU και TU. Για την Intra πρόβλεψη το H.264 διαθέτει μόλις 9 λειτουργείες κατεύθυνσης σάρωσης σε αντίθεση με το HEVC που διαθέτει 35 λειτουργείες. Επιπλέον και τα δύο πρότυπα διαθέτουν παρόμοιο φίλτρο deblocking με την διαφορά ότι στην περίπτωση του HEVC έχουν γίνει βελτιστοποιήσεις όσον αφορά τις τεχνικές παράλληλης επεξεργασίας. Για την κωδικοποίηση εντροπίας στο H.264 χρησιμοποιήθηκαν οι τεχνικές Context Adaptive Binary Arithmetic Coding (CABAC) και Context-Adaptive Variable-Length Coding (CAVLC), ενώ στο HEVC χρησιμοποιήθηκε μόνο η CABAC η οποία συνέβαλλε στην βελτίωση της ταχύτητας επεξεργασίας καθώς και στην αποδοτικότητα της κωδικοποίησης. Η αντιστάθμιση κίνησης στο H.264 γίνεται με την πρόβλεψη των διανυσμάτων κίνησης (Motion Vector Prediction) ενώ στο HEVC γίνεται με την πρόβλεψη εξειδικευμένων διανυσμάτων κίνησης (Advanced Motion Vector Prediction). Το HEVC χρησιμοποιεί ένα φίλτρο παρεμβολής 8-tap για την αντιστάθμιση κίνησης σε αντίθεση με το H.264 που χρησιμοποιεί 6-tap φίλτρο. Το tap ενός φίλτρου πεπερασμένης κρουστικής απόκρισης είναι μία τιμή συντελεστή και η κρουστική απόκριση του φίλτρου είναι οι συντελεστές του φίλτρου. Ο αριθμός των taps (N) είναι η ποσότητα μνήμης που απαιτείται για την εφαρμογή του φίλτρου. Συνεπώς το HEVC απαιτεί μεγαλύτερο εύρος ζώνης στην μνήμη για την κωδικοποίηση και αποκωδικοποίηση του βίντεο. Τέλος, το H.264 υποστηρίζει αναλύσεις μέχρι και 4K (3840 × 2160) με ρυθμό προβολής των frame 59.94 fps ενώ το HEVC υποστηρίζει αναλύσεις μεγαλύτερες από 8K (7680 × 4320) στα 300 fps.

4. Δομές Block στο HEVC

4.1. Coding Tree Units & Coding Tree Blocks

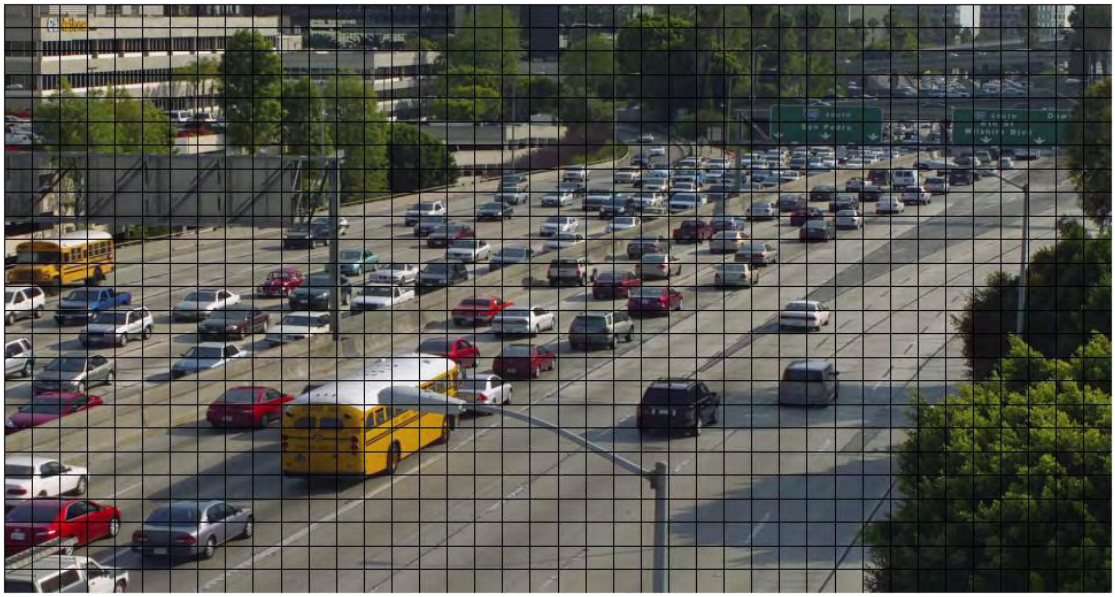
Σύμφωνα με όλα τα προηγούμενα πρότυπα κωδικοποίησης, κάθε εικόνα από ένα βίντεο χωρίζεται σε τετράγωνες δομές που λέγονται macroblock. Κάθε macroblock αποτελείται από ένα μπλοκ μεγέθους 16x16 που περιγράφει την τιμή της φωτεινότητας Y ή αλλιώς luma στον χρωματικό χώρο YUV και όταν χρησιμοποιείται η δειγματοληψία 4:2:0, που είναι η πιο διαδεδομένη ακολουθούν δυο μπλοκ μεγέθους 8x8, που περιγράφουν την τιμή του Cb και Cr αντίστοιχα. Για κάθε μακρομπλοκ πρέπει να ληφθεί η απόφαση για το τι είδους κωδικοποίηση θα γίνει στην πλευρά του κωδικοποιητή. Η απόφαση που θα ληφθεί επηρεάζει όλα τα στοιχεία του μακρομπλοκ καθορίζοντας αν σε αυτά θα γίνει intra πρόβλεψη ή πρόβλεψη με αντιστάθμιση κίνησης. Το μακρομπλόκ μεγέθους 16x16 είναι το μεγαλύτερο μέγεθος μπλοκ που μπορεί να χρησιμοποιηθεί για πρόβλεψη.

Παρ όλα αυτά το H.264/MPEG-4 AVC, πρόγονος του HEVC χρησιμοποιείται μέχρι και σήμερα για την αναπαραγωγή βίντεο υψηλής ευκρίνειας με τις διαδεδομένες αναλύσεις 1280x720 ή ακόμα και 1920x1080. Δεν σχεδιάστηκε όμως με αυτές τις προδιαγραφές. Το H.264 αρχικά σχεδιάστηκε για να επεξεργαστεί βίντεο ανάλυσης από QCIF 176x144 μέχρι SD 720x480 ή και 720x576 pixel [12]. Το HEVC σχεδιάστηκε να καλύψει τις ανάγκες που δημιουργήθηκαν από την διαδεδομένη χρήση βίντεο υψηλής ευκρίνειας, με αναλύσεις που φτάνουν τα 3840x2160 pixels. Σε αναλύσεις τέτοιου μεγέθους ο περιορισμός του μέγιστου μεγέθους ενός μπλοκ σε μόλις 16x16 pixels είναι μη αποδοτικός. Δηλαδή σε εικόνες υψηλής ευκρίνειας (HD) ή ακόμα και υπερ-υψηλής ευκρίνειας (UHD) πολλές περιοχές της εικόνας μπορούν να περιγραφούν με τις ίδιες παραμέτρους κίνησης είναι πολύ μεγαλύτερες από το μέγιστο μέγεθος του μακρομπλοκ 16x16 pixels. Η διαδικασία της σηματοδότησης για την εκάστοτε λειτουργεία κωδικοποίησης που θα οριστεί σε κάθε μακρομπλοκ μεγέθους 16x16 θα απαιτούσε ένα σημαντικό ποσοστό του bitrate.Επιπλέον λόγω της αυξημένης χωρικής συσχέτισης μεταξύ των γειτονικών δειγμάτων η χρήση μεγέθους μεγαλύτερου του 16x16 pixels είναι πλεονεκτική για αρκετά τμήματα της εικόνας. Η υποστήριξη μεγαλύτερων μεγεθών μπλοκ ήταν μία από τις βασικές πτυχές στις οποίες βασίστηκε το HEVC. Όπως αναφέρθηκε και νωρίτερα, η κάθε εικόνα διασπάται σε τετράγωνα μπλοκ, τα CTB, τα οποία περιγράφουν τα στοιχεία του χρωματικού χώρου YCbCr, δηλαδή το luma CTB, το Cb CTB και το Cr CTB. Κάθε μια τέτοια συστάδα αποτελεί το CTU (Coding Tree Unit). Το CTU αποτελεί την βασική μονάδα επεξεργασίας στο HEVC και είναι παρόμοια δομή με το macroblock. Στον χρωματικό χώρο YCbCr με δειγματοληψία 4:2:0 το luma CTB καλύπτει μια τετραγωνική περιοχή με μέγεθος $2^N * 2^N$ καθώς κάθε ένα από τα Cb & Cr μπλοκ καλύπτει μία τετραγωνική περιοχή μεγέθους $2^{N-1} * 2^{N-1}$. Η παράμετρος N προσδιορίζεται από τον κωδικοποιητή και μπορεί να πάρει τις τιμές N=4,5 και 6 , δημιουργώντας τα αντίστοιχα CTU με μέγεθος 16 x 16, 32 x 32 και 64 x 64. Συνήθως όσο μεγαλύτερο είναι το μέγεθος του CTU, τόσο πιο αποδοτική είναι η κωδικοποίηση. Το μεγαλύτερο

μέγεθος του CTU όμως έχει αντίκτυπο στην αύξηση του χρόνου κωδικοποίησης-αποκωδικοποίησης, αυξάνεται η κατανάλωση μνήμης και αυξάνεται η υπολογιστική πολυπλοκότητα της διαδικασίας της κωδικοποίησης.

A)

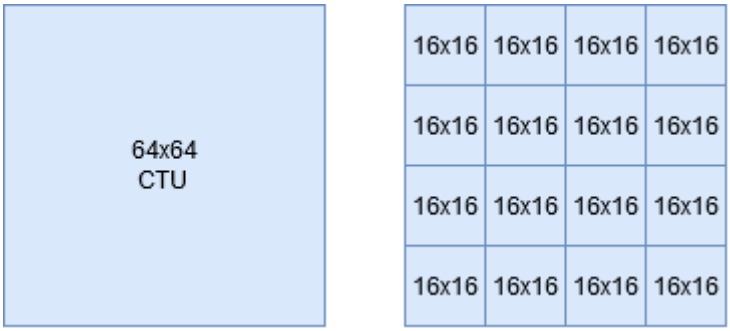
B)



Εικόνα 8: Παράδειγμα διάσπασης τις εικόνας μεγέθους 2560X1600 σε μακρομπλοκ και Coding Tree Units: Α) Διάσπαση σε μακρομπλοκ μεγέθους 16X16, τεχνική που εφαρμοζόταν μέχρι και στο H.264/MPEG-4 AVC.

Β) Διάσπαση της εικόνας σε 64X64 Coding Tree Units, το μεγαλύτερο μέγεθος το οποίο υποστηρίζει το H.265/HEVC.

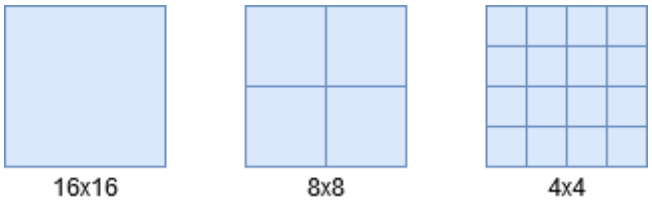
Στο παράδειγμα της εικόνας 8 παρατηρείται η ανάγκη χρήσης δομών μεγαλύτερων του 16 x 16 σε βίντεο υψηλής ανάλυσης καθώς πολλές περιοχές του βίντεο είναι πανομοιότυπες και θα μπορούσαν να περιγραφούν από ένα μόλις CTU αντί για 16 μακρομπλοκ.



Εικόνα 9: Ένα CTU μεγέθους 64x64 σε σύγκριση με το πλήθος των μακρομπλοκ μεγέθους 16x16 που απαιτούνται για να καλύψουν μια περιοχή μεγέθους 64x64

4.2. Coding Units & Coding Blocks

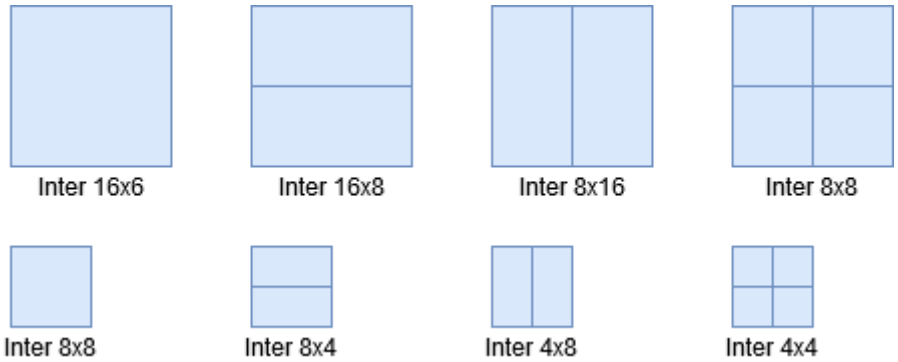
Σύμφωνα με τα προηγούμενα πρότυπα κωδικοποίησης το μακρομπλοκ είναι η μονάδα επεξεργασίας για την οποία επιλέγεται ένας τρόπος κωδικοποίησης από τον κωδικοποιητή. Για κάθε μακρομπλοκ αποφασίζεται εάν όλα τα luma και chroma δείγματα που περιλαμβάνονται σε αυτό μεταδοθούν χρησιμοποιώντας intra ή inter κωδικοποίηση. Το μακρομπλοκ μπορεί να διαιρεθεί σε μικρότερα μπλοκ με σκοπό την βελτιστοποίηση της πρόβλεψης. Το H.264/MPEG-4 AVC υποστηρίζει τρεις βασικές λειτουργείες για την Intra πρόβλεψη, οι οποίες είναι: η Intra 4x4 , η Intra 8x8 και η Intra 16x16 [12]. Στην Intra 4x4 το στοιχείο που περιγράφει την φωτεινότητα, luma διασπάται σε δεκαέξι μπλοκ μεγέθους 4x4. Τα στοιχεία κάθε μπλοκ προβλέπονται βάσει των γειτονικών ήδη κωδικοποιημένων μπλοκ. Στην Intra 8x8 κωδικοποίηση επαναλαμβάνεται η διαδικασία με την διαφορά ότι αντί για δεκαέξι μπλοκ μεγέθους 4x4 έχουμε τέσσερα μπλοκ μεγέθους 8x8. Στην λειτουργία 16x16 ολόκληρο το μπλοκ προβλέπεται χρησιμοποιώντας τα ήδη κωδικοποιημένα γειτονικά μπλοκ.



Εικόνα 10: Επιλογές διαμερίσεως ενός macroblock για την Intra πρόβλεψη που υποστηρίζει το H.264

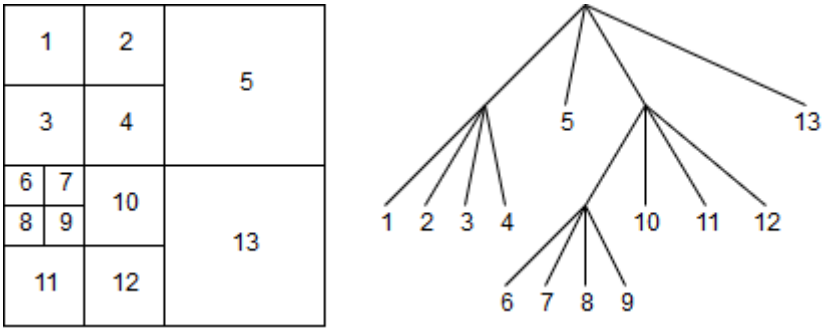
Όσον αφορά την Inter πρόβλεψη, το H.264 υποστηρίζει τις παρακάτω λειτουργίες: την Inter 16x16, την Inter 16x8, την Inter 8x16 καθώς και την Inter 8x8 [12]. Στην λειτουργία Inter 16x16 όλα τα luma και chroma δείγματα περιγράφονται από το ίδιο σύνολο των παραμέτρων κίνησης που προκύπτουν από την πρόβλεψη. Στις λειτουργείες των Inter 16x8 & Inter 8x16 τα luma και chroma μπλοκ που ανήκουν στο μακρομπλοκ χωρίζεται σε δυο οριζόντια και κάθετα ισομεγέθη ορθογώνια και κάθε ένα ορθογώνιο περιγράφεται από ξεχωριστό σύνολο παραμέτρων κίνησης. Στην λειτουργία Inter 8x8 το μακρομπλοκ χωρίζεται σε τέσσερα ισομεγέθη μπλοκ

μεγέθους 8x8. Κάθε μπλοκ μεγέθους 8x8 μπορεί να κωδικοποιηθεί ξεχωριστά ή ακόμα και να διασπαστεί εκ νέου σχηματίζοντας μπλοκ μεγέθους 8x4, 4x8 και 4x4.



Εικόνα 11: Επιλογές διαμερίσεως ενός macroblock για Inter πρόβλεψη που υποστηρίζονται από το H.264

Σύμφωνα με τις προδιαγραφές του HEVC η απόφαση για το τι είδους πρόβλεψη θα χρησιμοποιηθεί μπορεί να ληφθεί στο επίπεδο του CTU, δηλαδή σε ένα μπλοκ που έχει μέγεθος έως και 64x64. Η άμεση εφαρμογή των διαδικασιών που περιεγράφηκαν παραπάνω, η μεταφορά δηλαδή από το πεδίο του macroblock στο πεδίο του CTU θα είχε σημαντικό κόστος απώλειας ποιότητας. Για την επίλυση αυτού του προβλήματος προστέθηκε ακόμα μια δομή στο HEVC, το Coding Unit (CU). Το Coding Tree Unit μπορεί να διασπαστεί σε πολλαπλά CUs, με την δυνατότητα να λαμβάνουν μεταβλητά μεγέθη, από 8x8 έως και 64x64. Η δυνατότητα αυτή δημιούργησε την ανάγκη για μία δομή η οποία θα περιγράφει την δομή του CTU, αυτή είναι η δομή τετραδικού δέντρου όπως φαίνεται και την Εικόνα 12.



Εικόνα 12: Ένα CTU διασπασμένο σε CUs και η δομή τετραδικού δέντρου.

Τα CUs που αποτελούν το CTU κωδικοποιούνται με προτεραιότητα κατά βάθος (depth-first order), αυτή η προσέγγιση είναι γνωστή και ως z-scan [12]. Το μήκος και το πλάτος του βίντεο θα πρέπει να είναι πολλαπλάσια του ελάχιστου μεγέθους που μπορεί να λάβει ένα CU. Δεν είναι όμως υποχρεωτικό να είναι πολλαπλάσιο του μεγέθους του CTU, καθώς εάν στην περίπτωση που δεν είναι ακέραιο πολλαπλάσιο του CTU, τότε το CTU διασπάται σε CUs μέχρι αυτά να εφάπτονται στα άκρα της εικόνας. Το CU πλέον αποτελεί την βασική μορφή στο επίπεδο της οποίας λαμβάνεται η απόφαση από τον κωδικοποιητή εάν θα χρησιμοποιηθεί Inter ή Intra πρόβλεψη που αφορά τα luma και chroma μπλοκ, τα οποία απαρτίζουν το CU. Πλέον

η έννοια του CU μπορεί να συσχετιστεί με αυτή του μακρομπλοκ, που εφαρμόστηκε στα προηγούμενα πρότυπα κωδικοποίησης, μέχρι το H.264. Διαφέρουν όμως αρκετά καθώς τα CUs έχουν μεταβλητά μεγέθη, επίσης τα CUs υπόκεινται σε περεταίρω διάσπαση σχηματίζοντας δομές όπως το Transform Unit (TU) ή το Prediction Unit (PU) τα οποία θα αναπτυχθούν παρακάτω. Το βασικό πλεονέκτημα της χρήσης της δομής κωδικοποίησης δέντρου είναι η επίτευξη μίας ενοποιημένης δομής προσδιορισμού της κατανομής των CTU σε μπλοκ που χρησιμοποιούνται για την Inter, την Intra πρόβλεψη καθώς και την κωδικοποίηση μετασχηματισμού.

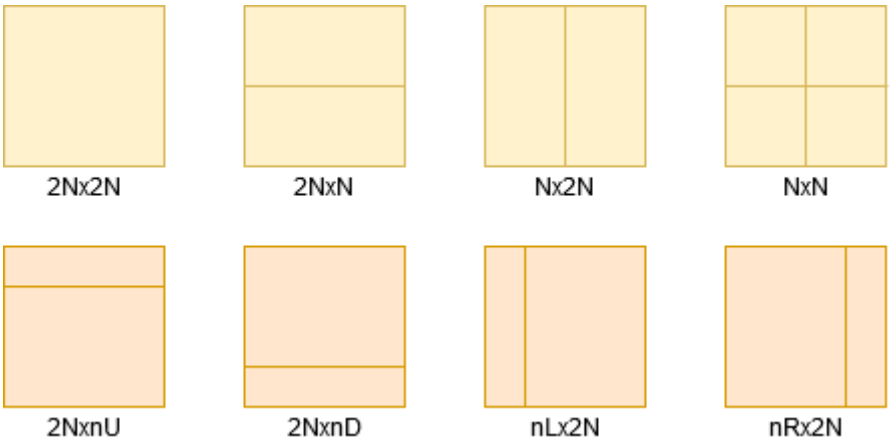
4.3. Prediction Blocks & Prediction Units

Για κάθε CU επιλέγεται η μορφή πρόβλεψης με την οποία θα κωδικοποιηθεί, δηλαδή Inter ή Intra πρόβλεψη. Εάν το CU έχει το μικρότερο επιτρεπτό μέγεθος που ορίζεται από τις παραμέτρους κωδικοποίησης, τότε το luma CB μπορεί να διασπαστεί σε τέσσερα ισομεγέθη τετράγωνα μπλοκ και κάθε ένα από αυτά μπορεί να κωδικοποιηθεί με διαφορετική λειτουργία όταν είναι επιλεγμένη η Intra πρόβλεψη. Η Intra πρόβλεψη δεν εφαρμόζεται πάντα στα μπλοκ που σηματοδοτήθηκαν για την πρόβλεψη αυτή. Ένα CB μπορεί να διασπαστεί σε πολλαπλά μπλοκ μετασχηματισμού (Transform Block ή TB), που αναπαριστούν τις οντότητες στις οποίες λαμβάνει θέση ένας δισδιάστατος μετασχηματισμός για την κωδικοποίηση των υπολειμμάτων της πρόβλεψης. Όταν ένα luma CB διασπάται σε τέσσερα μπλοκ για την σηματοδότηση των λειτουργιών Intra πρόβλεψης, διασπάται επίσης για τον σκοπό της κωδικοποίησης μετασχηματισμού, έτσι ώστε όλα τα δείγματα μέσα στο TB να προβλέπονται χρησιμοποιώντας την ίδια λειτουργία Intra πρόβλεψης.

Εάν ένα για ένα CU επιλεχθεί η Inter πρόβλεψη τα luma και chroma CB μπορούν να διασπαστούν σε Prediction Blocks. Ένα PB είναι ένα μπλοκ που περιγράφει το luma στοιχείο ή ένα chroma στοιχείο που περιγράφεται από το ίδιο σύνολο παραμέτρων κίνησης. Στα δυο chroma CBs, το Cb και το Cr ενός CU εφαρμόζεται η ίδια διάσπαση που εφαρμόστηκε και στο luma CB. Το luma PB μαζί με τα δυο chroma PBs αποτελούν ένα Prediction Unit (PU).

Το HEVC υποστηρίζει οχτώ τρόπους διαμέρισης ενός CU σε PUs [11]. Ένα CU μπορεί να κωδικοποιηθεί ως ένα μόνο PU, ή μπορεί να διασπαστεί σε δύο ορθογώνια PUs συμμετρικά καθώς και μη ή ακόμα και σε τέσσερα ισομεγέθη τετράγωνα. Στην περίπτωση όπου ολόκληρο το CU κωδικοποιείται ως ένα μόνο PU τότε αυτό είναι της μορφής 2Nx2N. Στην περίπτωση που το CU διασπάται σε τέσσερα ισομεγέθη PUs, αυτά είναι της μορφής NxN και ισχύει ένας περιορισμός, για να γίνει αυτή η διάσπαση το CU θα πρέπει να βρίσκεται στο μικρότερο δυνατό μέγεθος που ορίζεται από τις παραμέτρους κωδικοποίησης. Για την διάσπαση του CU σε δυο ισομεγέθη ορθογώνια PUs παρατηρείται η μορφή 2NxN καθώς και η Nx2N, που είναι διασπασμένα κάθετα και οριζόντια αντίστοιχα. Επίσης το CU μπορεί να διασπαστεί και σε μη συμμετρικά ορθογώνια PUs, που διακρίνονται σε δύο κάθετα και δύο

οριζόντια διασπασμένα ορθογώνια. Όσο για τα κάθετα, τόσο και για τα οριζόντια διασπασμένα ορθογώνια στην μία περίπτωση το ένα μπλοκ καλύπτει το ¼ του μεγέθους του ολικού μπλοκ καθώς το υπόλοιπο ορθογώνιο καλύπτει τα ¾ αυτού. Εδώ συναντώνται οι μορφές nLx2N και nRx2N όπου L και R σημαίνουν αριστερά και δεξιά αντίστοιχα για τα κάθετα διασπασμένα ορθογώνια. Στα οριζόντια διασπασμένα ορθογώνια συναντώνται οι μορφές 2NxnU και 2NxnD όπου U και D αντιστοιχούν στο πάνω τμήμα και κάτω τμήμα αντίστοιχα. Η διάσπαση σε μη συμμετρικά μπλοκ υποστηρίζεται για CU μεγέθους 8x8 και άνω.



Εικόνα 13: Οι μορφές διάσπασης που μπορεί να λάβει το Prediction Unit

4.4. Transform Blocks & Transform Units

Αφού πραγματοποιηθεί η πρόβλεψη, πρέπει να κωδικοποιηθούν οι διαφορές, δηλαδή η διαφορά μεταξύ της πρόβλεψης και της αρχικής εικόνας. Αυτό δεν μπορεί να γίνει πάντα στο επίπεδο του CB, καθώς είναι τέτοιο το μέγεθός του που μπορεί να οδηγήσει σε απώλεια πληροφορίας η πρόβλεψη. Έτσι το CB μπορεί να διασπαστεί σε Transform Blocks (TB). Ένα TB αναπαριστά ένα τετράγωνο μπλοκ αποτελούμενο από δείγματα που περιγράφουν ένα τμήμα της εικόνας, κοινώς luma η chroma. Η διάσπαση ενός CB σε TB γίνεται αναδρομικά ακολουθώντας μία δομή τετραδικού δέντρου το οποίο έχει ρίζα το CB και ονομάζεται Residual Quadtree (RQT).

5. HEVC VIDEO ANALYSIS TOOL

Σε αυτό το κεφάλαιο θα γίνει λεπτομερής περιγραφή της εφαρμογής που αναπτύχθηκε, το "HEVC Video Analysis Tool" καθώς και των μέσων που χρησιμοποιήθηκαν για την ανάπτυξη αυτής. Επίσης το κεφάλαιο αυτό έχει και την ιδιότητα του οδηγού για κάποιον που θέλει να χρησιμοποιήσει την εφαρμογή για να κατανοήσει πλήρως τις λειτουργίες αυτής. Το "HEVC Video Analysis Tool" όπως αναφέρθηκε και στην εισαγωγή έχει ήδη υλοποιηθεί στην βασική του μορφή. Σκοπός της παρούσας διπλωματικής εργασίας είναι η προσθήκη λειτουργειών, οι οποίες θα δώσουν στον χρήστη παραπάνω δυνατότητες και πληροφορίες. Επίσης έγιναν βελτιστοποιήσεις όσον αφορά το γραφικό περιβάλλον χρήστη (GUI) καθώς και την αλγοριθμική και συνεπώς χρονική πολυπλοκότητα της εφαρμογής.

Αρχικά, η εφαρμογή αναπτύχθηκε χρησιμοποιώντας τις γλώσσες προγραμματισμού C και C++. Καθ' όλη τη διάρκεια της ανάπτυξης χρησιμοποιήθηκε το ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) Microsoft Visual Studio 2019. Η εφαρμογή στην παρούσα υλοποίηση είναι συμβατή μόνο με το λειτουργικό σύστημα Windows.

Οι ενέργειες που έγιναν για να επιτευχθεί το αποτέλεσμα αυτό μπορούν να κατηγοριοποιηθούν σε τρείς ομάδες. Στο πρώτο στάδιο πρέπει να τροποποιηθεί η εφαρμογή του αποκωδικοποιητή, έτσι ώστε να είμαστε σε θέση να αποσπάσουμε τα απαραίτητα δεδομένα, τα οποία προκύπτουν κατά την διάρκεια της αποκωδικοποίησης, απαραίτητα για το επόμενο στάδιο και η αποθήκευση αυτών. Στην συνέχεια, δεύτερο στάδιο, τροποποιείται η εφαρμογή "HEVC Video Analysis Tool". Αναπτύσσεται αλγόριθμος, σκοπός του οποίου είναι η είσοδος των δεδομένων που ανακτήθηκαν από το προηγούμενο στάδιο στην εφαρμογή και η αποθήκευσή τους σε απαραίτητες δομές που δημιουργήθηκαν για το επόμενο στάδιο. Τέλος, το τρίτο στάδιο περιλαμβάνει την τροποποίηση της εφαρμογής "HEVC Video Analysis Tool", για να επιτευχθεί η οπτικοποίηση των δεδομένων που αντλήθηκαν και επεξεργάστηκαν στα προηγούμενα στάδια.

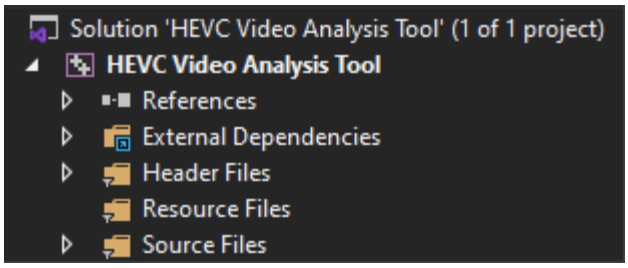
Το ζεύγος κωδικοποιητή – αποκωδικοποιητή (codec) που χρησιμοποιήθηκε για την εκπόνηση της παρούσας εργασίας, καθώς και τις δοκιμές είναι το HEVC Test Model (HM) Reference Software, το οποίο αναπτύχθηκε από την Joint Collaborative Team on Video Coding (JCT-VC), η οποία προέκυψε από την συνεργασία της ITU-T, μέλος του Video Coding Experts Group (VCEG) και του ISO/IEC, μέλη του Motion Pictures Experts Group (MPEG). Ο κωδικοποιητής χρησιμοποιήθηκε για τον έλεγχο της ορθότητας των δεδομένων όπου έπειτα από επεξεργασία εντός της εφαρμογής παρουσιάζονται στον χρήστη. Αρχικά, διαθέτουμε ένα βίντεο σε ασυμπίεστη μορφή (file_to_be_encoded.yuv), την εφαρμογή του κωδικοποιητή καθώς και ένα αρχείο ρυθμίσεων (profile.cfg). Το "profile.cfg" δίνει την δυνατότητα στον χρήστη να μεταβάλλει τις παραμέτρους κωδικοποίησης έτσι ώστε να έχει το επιθυμητό αποτέλεσμα. Εφ' όσον ολοκληρωθεί η διαδικασία της κωδικοποίησης, έχουμε στην

διάθεσή μας δύο αρχεία, το κωδικοποιημένο βίντεο σε .bin μορφή καθώς και το ανακατασκευασμένο αρχείο σε .yuv μορφή. Το αρχείο που θα χρησιμοποιηθεί στην συνέχεια είναι το κωδικοποιημένο το οποίο έχει κατάληξη .bin.

5.1. Το περιβάλλον ανάπτυξης

Η διαδικασία ανάπτυξης κώδικα έγινε με την χρήση του ολοκληρωμένου περιβάλλοντος ανάπτυξης Microsoft Visual Studio 2019 Community Edition. Το Visual Studio επιλέχθηκε καθώς αποτελεί μια ολοκληρωμένη λύση για τις απαιτήσεις της υλοποίησης της εφαρμογής. Κάποια σημαντικά στοιχεία που περιέχει για τις απαιτήσεις της εργασίας αυτής είναι τα παρακάτω: ενσωματωμένο compiler, debugger, code editor και σχεδιαστής γραφικού περιβάλλοντος.

Στο Microsoft Visual Studio η ιεραρχία οργάνωσης των αρχείων της εφαρμογής που αναπτύσσεται έχει την ακόλουθη μορφή (Εικόνα 14). Solution είναι η δομή που χρησιμοποιείται για την οργάνωση των projects στο Visual Studio. Μπορεί να περιέχει πάνω από ένα projects, όπου στην περίπτωση του παραδείγματος της Εικόνας 14 υπάρχει μόνο ένα project. Κάθε project αποτελείται από πέντε διαφορετικές οντότητες, όπου η κάθε μια έχει συγκεκριμένο ρόλο και παρέχει τις ανάλογες πληροφορίες για την μεταγλώττιση του πηγαίου κώδικα σε εκτελέσιμο αρχείο. Στο περιβάλλον του Microsoft Visual Studio ένα project μπορεί να είναι ένα εκτελέσιμο αρχείο (.exe), μια βιβλιοθήκη δυναμικής σύνδεσης (.dll), μια στατική βιβλιοθήκη (.lib) ή ακόμα και ένα makefile project. Το makefile είναι ένα αρχείο που περιέχει κανόνες όπου βάσει αυτών γίνεται η μεταγλώττιση από πηγαίο κώδικα σε κώδικα μηχανής.



Εικόνα 14: Ιεραρχία οργάνωσης των αρχείων στο Microsoft Visual Studio

- **References:** Όταν γράφεται κώδικας που συνδέει ένα εξωτερικό στοιχείο ή μια υπηρεσία πρέπει να γραφεί η αναφορά σε αυτή. Μια αναφορά είναι ουσιαστικά μια καταχώρηση σε ένα αρχείο ενός project που περιέχει τις πληροφορίες που χρειάζεται το Visual Studio για να εντοπίσει το στοιχείο ή την υπηρεσία.
- **External Dependencies:** Τα External Dependencies ή αλλιώς εξωτερικές εξαρτήσεις περιέχουν όλες τις βιβλιοθήκες που είναι εγκατεστημένες στον

υπολογιστή του χρήστη καθώς και απαραίτητες για την ανάπτυξη της εφαρμογής.

- **Header Files:** Header File ή αλλιώς αρχείο κεφαλίδας είναι ένα αρχείο που έχει την κατάληξη .h, χρησιμοποιείται από τις C/C++ και περιέχει την δήλωση μεθόδων ή μεταβλητών και ορισμό μακροεντολών για κοινή χρήση μεταξύ αρχείων πηγαίου κώδικα. Για να μπορέσουν να χρησιμοποιηθούν τα αρχεία κεφαλίδας πρέπει στο συμπεριληφθούν στο αρχείο πηγαίου κώδικα, αυτό γίνεται γράφοντας στην αρχή του αρχείου με κατάληξη .c ή .cpp #include και το όνομα του αρχείου. Επιπλέον, υπάρχουν δύο είδη αρχείων κεφαλίδας, αυτά που συνοδεύουν τον compiler καθώς και αυτά που γράφει ο χρήστης. Στην πρώτη περίπτωση το include συντάσσεται με την μορφή #include <header_file.h> , ενώ στην περίπτωση που το αρχείο κεφαλίδας το γράφει ο χρήστης, τότε το include συντάσσεται με την παρακάτω μορφή, #include “header_file.h”.
- **Resource Files:** Είναι μια κατηγορία αρχείων που μπορεί να έχει πολλές υποστάσεις, για παράδειγμα μπορεί να είναι ένα resource script με την κατάληξη .rc, μπορεί να είναι ένα αρχείο προτύπου πόρων με την κατάληξη .rct , ένας μεμονωμένος πόρος, για παράδειγμα ένα εικονίδιο ή ένας κέρσορας. Ακόμα θα μπορούσε να είναι κάποιο αρχείο κεφαλίδας που δημιουργήθηκε από το περιβάλλον ανάπτυξης, για παράδειγμα ένα αρχείο resource.h το οποίο αναφέρεται στο αρχείο .rc. Οι πόροι μπορούν ακόμα να έχουν την μορφή εκτελέσιμου αρχείου (.exe) ή βιβλιοθήκης δυναμικής σύνδεσης.
- **Source Files:** Τα πηγαία αρχεία περιέχουν τον πηγαίο κώδικα του project καθώς και βασικές κλάσεις που χρησιμοποιούνται από το project και τις υλοποιήσεις αυτών. Περιλαμβάνουν την ανάπτυξη μεθόδων που ορίστηκαν στα αρχεία κεφαλίδας καθώς και το σημείο έναρξης της εφαρμογής

5.2. OpenCV

Η Open Source Computer Vision Library είναι μια βιβλιοθήκη λογισμικού υπολογιστικής όρασης και μηχανικής μάθησης ανοιχτού κώδικα. Όντας προϊόν με άδεια BSD, η OpenCV διευκολύνει τον χρήστη να χρησιμοποιεί και να τροποποιεί τον κώδικα. Η βιβλιοθήκη διαθέτει περισσότερους από 2500 βελτιστοποιημένους αλγόριθμους, οι οποίοι περιλαμβάνουν ένα ολοκληρωμένο σύνολο κλασικών και προηγμένων αλγορίθμων υπολογιστικής όρασης και μηχανικής μάθησης. Η OpenCV έχει γραφεί σε C++ καθώς υπάρχουν και βιβλιοθήκες που υποστηρίζουν γλώσσες όπως η Python, η Java και Matlab. Η OpenCV χρησιμοποιήθηκε στην παρούσα εργασία για την επίτευξη της οπτικοποίησης και αναπαραγωγής μιας ακολουθίας βίντεο. Πιο συγκεκριμένα χρησιμοποιήθηκαν οι παρακάτω μέθοδοι.

- `IplImage* cvCreateImage(CvSize size, int depth, int channels)`
  Δημιουργεί την κεφαλίδα της εικόνας και δεσμεύει την μνήμη που απαιτείται.

Έχει ως ορίσματα το μέγεθος της εικόνας σε μια δομή δεδομένων που ονομάζεται `cvSize(int width, int height)` και έχει ως ορίσματα το πλάτος και το ύψος της εικόνας. Επίσης έχει ως όρισμα το χρωματικό βάθος της εικόνας για την οποία δεσμεύεται μνήμη καθώς και τον αριθμό των καναλιών ανά pixel.

- `void cvtColor(InputArray src, OutputArray dst, int code)`
  Μεταφέρει την εικόνα από τον ένα χρωματικό χώρο στον άλλο. Έχει ως ορίσματα την εικόνα προς μετατροπή, την εικόνα που προκύπτει από την μεταφορά από τον ένα χρωματικό χώρο στον άλλο με το ίδιο μέγεθος και βάθος χρώματος καθώς και τον κωδικό μετατροπής χρωματικού χώρου. Στην συγκεκριμένη υλοποίηση χρησιμοποιήθηκε ο κωδικός μεταφοράς από τον χρωματικό χώρο YCrCb στον RGB.

- `Mat cvarrToMat(const CvArr* arr, bool copyData=false, bool allowND=true, int coiMode=0 )`
  Η μέθοδος αυτή μετατρέπει μια εικόνα σε ένα αντικείμενο της κλάσης `Mat` και επιστρέφει το αντικείμενο αυτό. Η κλάση `Mat` αντιπροσωπεύει έναν πίνακα Ν διαστάσεων ενός καναλιού ή παραπάνω.

- `int waitKey(int delay=0)`
  Η μέθοδος αυτή χρησιμοποιήθηκε για τον χειρισμό των events στην γραφική διεπαφή χρήστη που σχεδιάστηκε. Όταν χρησιμοποιείται ως όρισμα το `delay=0` η εφαρμογή αναπαράγει μια εικόνα μέχρι να λάβει μέρος κάποιο event.

5.3. Κώδικας και οι μέθοδοι που αναπτύχθηκαν

5.3.1. Decoder

Όπως αναφέρθηκε προηγουμένως, αφού ο χρήστης επιλέξει την ακολουθία που επιθυμεί να αναλύσει, το HEVC Video Analysis Tool καλεί για εκτέλεση τον αποκωδικοποιητή προκειμένου να αποσυμπιεστεί η ακολουθία και να προβληθεί. Για να επιτευχθούν οι στόχοι της παρούσας εργασίας πρέπει κατά την διάρκεια της αποκωδικοποίησης να ανακτηθεί πλήθος πληροφοριών που προκύπτουν κατά την διάρκεια της αποκωδικοποίησης και να αποθηκευτούν στην συνέχεια. Για αυτόν τον λόγο τροποποιήθηκε η εφαρμογή του codec. Το HM reference software version 16.15 είναι γραμμένο στο περιβάλλον του Microsoft Visual Studio και παρέχει το αρχείο HM_vc2015.sln το οποίο περιγράφει το Solution. Το συγκεκριμένο Solution αποτελείται από εννέα project τα οποία διακρίνονται σε εκτελέσιμα αρχεία, είτε σε στατικές βιβλιοθήκες. Στο Solution προστέθηκε ακόμα ένα project με την ιδιότητα

της στατικής βιβλιοθήκης (.lib) το οποίο ονομάστηκε Parser. Το project αυτό αποτελείται από πέντε αρχεία κεφαλίδας, τα οποία είναι τα παρακάτω.

- **CtuParser.h:** Το οποίο δημιουργεί ένα αρχείο κειμένου (.txt) και καταγράφει πληροφορίες σχετιζόμενες με το Coding Unit που αποσυμπιέζεται κατά την διάρκεια της αποκωδικοποίησης. Πιο συγκεκριμένα, καταγράφει πληροφορίες όπως: Το frame στο οποίο ανήκει, το βάθος του CU καθώς και τις συντεταγμένες των τεσσάρων σημείων που ορίζουν την τετράγωνη περιοχή του CU.
- **Ctu2Parser.h:** Δημιουργεί επίσης ένα αρχείο κειμένου και καταγράφει πληροφορίες σχετιζόμενες με το Coding Unit. Καταγράφει το frame στο οποίο ανήκει, τον τρόπο με τον οποίο έγινε η πρόβλεψη, δηλαδή inter ή intra, τις διαστάσεις του, δηλαδή μήκος και πλάτος καθώς και εάν έχει γίνει merge.
- **SliceParser.h:**Δημιουργεί ένα αρχείο κειμένου και καταγράφει πληροφορίες που περιγράφουν την δομή του Slice. Πιο συγκεκριμένα καταγράφει το frame στο οποίο ανήκει το Slice, την διεύθυνση του πρώτου CTU του Slice καθώς και τον συνολικό αριθμό των CTU που υπάρχουν στο frame πριν ξεκινήσει το τρέχον Slice.
- **SpsParser.h:** Δημιουργεί ένα αρχείο κειμένου και καταγράφει πληροφορίες που μεταφέρονται από το Sequence Parameter Set. Πιο συγκεκριμένα περιγράφει τις διαστάσεις του βίντεο σε μήκος και πλάτος, το χρωματικό βάθος με το οποίο έχει καταγραφεί το βίντεο, τον τύπο της δειγματοληψίας χρώματος σύμφωνα με την οποία έχει καταγραφεί το βίντεο, το μέγιστο μέγεθος που μπορεί να λάβει ένα CU, το μέγιστο βάθος που μπορεί να έχει ένα CU καθώς και τα μέγιστα βάθη των τετραδικών δέντρων που περιγράφουν την δομή TU και αφορούν Intra ή Inter κωδικοποιημένα TU.
- **TileParser.h:** Καταγράφει σε ένα αρχείο κειμένου τον αριθμό του frame στο οποίο βρίσκεται το Tile, τον συνολικό αριθμό των CTU που βρίσκονται εντός του Tile καθώς και την διεύθυνση του πρώτου CTU του Tile.

Τα παραπάνω αρχεία κεφαλίδας περιέχουν μεθόδους που καλούνται από μεθόδους που βρίσκονται εντός των κλάσεων του project του αποκωδικοποιητή. Οι μέθοδοι αυτές καλούνται αναδρομικά με αποτέλεσμα να γίνονται εκατοντάδες κλήσεις αυτών για κάθε frame του βίντεο. Αυτό που παρατηρήθηκε ήταν ότι κάθε φορά που καλούνταν η μέθοδος δημιουργούνταν ένα νέο object αυτής με αποτέλεσμα στις εκατοντάδες φορές που καλείται η μέθοδος αυτή να ανοίγει το αρχείο πριν την εγγραφή και να κλείνει στην συνέχεια για κάθε κλήση της, με συνέπεια η διαδικασία της αποκωδικοποίησης να γίνεται χρονοβόρα.

Για την επίλυση του παραπάνω προβλήματος επιλέχθηκε το μοτίβο ανάπτυξης κώδικα που ονομάζεται Singleton. Το Singleton είναι ένα μοτίβο ανάπτυξης λογισμικού το οποίο επιτρέπει σε μια κλάση να έχει ένα και μόνο ένα instance. Το Singleton ουσιαστικά είναι μία τεχνική ανάπτυξης σύμφωνα με την οποία ο

constructor της κλάσης γίνεται private έτσι ώστε να μην υπάρχει πρόσβαση εξωτερικά της κλάσης καθώς και δημιουργείται μια στατική μέθοδος που ονομάστηκε getInstance() στην συγκεκριμένη υλοποίηση η οποία είναι υπεύθυνη για τον έλεγχο εάν υπάρχει ήδη κάποιο instance της κλάσης ή όχι. Εάν υπάρχει τότε επιστρέφει το υπάρχον instance αλλιώς δημιουργεί νέο instance. Με αυτόν τον τρόπο το αρχείο καταγραφής ανοίγει μόνο μια φορά, όταν καλείται ο constructor, παραμένει ανοιχτό και κλείνει με την αποπεράτωση της διαδικασίας αποκωδικοποίησης.

```
class SingletonClass
{
public:

    SingletonClass() {
        //Costructor Code..
    }
    ~SingletonClass()
    {
        //Destructor Code..
    }
    void SingletonClass::SingletonMethod()
    {
        //Method Code..
    }

    static SingletonClass* getInstance()
    {
        if (m_instance == NULL)
            m_instance = new SingletonClass;
        return m_instance;
    }
private:
    //Private variable Declaration..
    static SingletonClass* m_instance;
};
SingletonClass* SingletonClass::m_instance = NULL;
```

Snippet 1 Το μοτίβο ανάπτυξης Singleton

```
SingletonClass::getInstance()->SingletonMethod();
```

Snippet 2 Κλήση μεθόδου που ανήκει σε κλάση όπου έχει χρησιμοποιηθεί Singleton

5.3.2. Εφαρμογή

Η εφαρμογή HEVC Video Analysis Tool αποτελείται από πέντε αρχεία κεφαλίδας και αντίστοιχα πέντε αρχεία πηγαίου κώδικα και είναι τα παρακάτω:

- MainForm.h
- PsnrForm.h
- Psnr.h
- VideoYUV.h
- Yuv.h
- MainForm.cpp
- PsnrForm.cpp
- Psnr.cpp
- VideoYUV.cpp
- Yuv.cpp

Αποτέλεσμα της μεταγλώττισης των παραπάνω αρχείων είναι η εκτελέσιμη εφαρμογή που περιλαμβάνει γραφική διεπαφή χρήστη. Για την ακρίβεια, τα αρχεία MainForm.h και PsnrForm.h παρόλο που έχουν την επέκταση .h δεν θεωρούνται αρχεία κεφαλίδας από το Microsoft Visual Studio αλλά αποτελούν αρχεία φόρμας C++ (C++ Form File). Τα αρχεία MainForm.h και PsnrForm.h περιγράφουν τα δυο παράθυρα γραφικών που περιλαμβάνει η εφαρμογή. Τα παράθυρα γραφικών σχεδιάστηκαν με την βοήθεια του εργαλείου σχεδιασμού που παρέχει το Microsoft Visual Studio. Στην συνέχεια θα περιγραφούν αναλυτικά οι τεχνικές λεπτομέρειες, οι δομές δεδομένων που χρησιμοποιήθηκαν καθώς και οι μέθοδοι που περιλαμβάνονται στα παραπάνω αρχεία.

Το αρχείο MainForm.h περιγράφει το βασικό παράθυρο της εφαρμογής, τα στοιχεία που περιέχει αυτό καθώς και τις οδηγίες για τον χειρισμό των ενεργειών που προκύπτουν από την αλληλεπίδραση του χρήστη με τα στοιχεία που περιλαμβάνει το παράθυρο αυτό. Το αρχείο αυτό περιλαμβάνει κώδικα γραμμένο σε γλώσσες C, C++ και C++/CLI. Οι γλώσσες αυτές αναγνωρίζονται από τον μεταγλωττιστή μέσω του Common Language Runtime (CLR). Όντας αρχείο φόρμας υπάρχουν τρία διαθέσιμα περιβάλλοντα εργασίας, σε αντίθεση με τα αρχεία κεφαλίδας. Το περιβάλλον σχεδίασης της φόρμας, το περιβάλλον ανάπτυξης κώδικα καθώς και το διάγραμμα κλάσεων, το οποίο δεν χρησιμοποιήθηκε στην παρούσα υλοποίηση. Το περιβάλλον σχεδίασης δίνει στον προγραμματιστή την δυνατότητα να σχεδιάσει και να αναπτύξει το γραφικό περιβάλλον της φόρμας αξιοποιώντας την γραφική διεπαφή που παρέχει το Microsoft Visual Studio. Ουσιαστικά διευκολύνεται η διαδικασία ανάπτυξης του προτύπου της φόρμας καθώς μπορούν να προστεθούν στοιχεία όπως κουμπιά, μενού, πλαίσια κειμένου και πλαίσια αναπαραγωγής εικόνας στην φόρμα και να παραμετροποιηθούν εύκολα. Το περιβάλλον ανάπτυξης κώδικα δίνει την δυνατότητα στον προγραμματιστή να δηλώσει μεταβλητές και μεθόδους, να αναπτύξει τις μεθόδους αυτές και να αναπτύξει τμήματα κώδικα προκειμένου να καθορίσουν την λειτουργικότητα της εφαρμογής όταν αλληλεπιδρά με τον χρήστη, για παράδειγμα το πάτημα ενός κουμπιού. Κατά την διάρκεια της επεξεργασίας μίας φόρμας στο περιβάλλον σχεδίασης δημιουργείται αυτόματα κώδικας που περιγράφει δομές και μεταβλητές της φόρμας και των περιεχομένων της.

Το αρχείο MainForm.h αρχικά περιλαμβάνει τις δηλώσεις των απαιτούμενων βιβλιοθηκών έτσι ώστε να συμπεριληφθούν κατά την διάρκεια της μεταγλώττισης. Κάποιες από αυτές παρέχονται από την υλοποίηση του μεταγλωττιστή, κάποιες από αυτές έχουν εγκατασταθεί στον υπολογιστή αργότερα και κάποιες από αυτές έχουν γραφτεί ή τροποποιηθεί από εμένα. Οι βιβλιοθήκες που συμπεριλαμβάνονται στην εφαρμογή είναι οι παρακάτω: Η OpenCV η οποία χρησιμοποιήθηκε για την ανάγνωση των ακολουθιών βίντεο και την αναπαραγωγή τους, η omp.h η οποία είναι βιβλιοθήκη παράλληλου προγραμματισμού και επιτρέπει την δημιουργία και τον έλεγχο διεργασιών εντός της εφαρμογής, η string.h και η msclr\marshal.h ,βιβλιοθήκες που περιλαμβάνουν μεθόδους χειρισμού και επεξεργασίας των

συμβολοακολουθιών, η PsnrForm.h και η Psnr.h όπου αποτελούν μια φόρμα και ένα αρχείο κεφαλίδας αντίστοιχα, τα οποία περιλαμβάνουν δομές που είναι απαραίτητες για τον υπολογισμό του PSNR. Τέλος συμπεριλαμβάνονται τα αρχεία κεφαλίδας yuv.h και VideoYUV.h τα περιέχουν δομές και μεθόδους απαραίτητες για την ανάγνωση των ακολουθιών βίντεο.

Το επόμενο τμήμα του κώδικα περιέχει την δήλωση και ανάπτυξη των δομών δεδομένων που δημιουργήθηκαν.

- **Struct frame:** Χρησιμοποιήθηκε για την αποθήκευση δεδομένων που προέκυψαν κατά την αποκωδικοποίηση και την οπτικοποίηση αυτών. Αποτελείται από μεταβλητές ακέραιου τύπου οι οποίες περιγράφουν ένα frame καθώς και από δείκτες που δείχνουν σε άλλες δομές.
    - *Int frameWidth:* Το πλάτος του frame σε δείγματα luma.
    - *Int frameHeight:* Το ύψος του frame σε δείγματα luma.
    - *Int numOfSlices:* Ο αριθμός των slices που περιέχει ένα frame.
    - *Int numOfTiles:* Ο αριθμός των tiles που περιέχει ένα frame.
    - *Int numOfCUs:* Ο αριθμός των CUs που περιέχει ένα frame.
    - *Int widthInCTUs:* Το πλάτος του frame σε CTUs.
    - *Int heightInCTUs:* Το πλάτος του frame σε CTUs.
    - *Int CTUwidth:* Το πλάτος του CTU σε δείγματα luma.
    - *Int CTUheight:* Το ύψος του CTU σε δείγματα luma.
    - *struct slice* sl:* Δείκτης που περιγράφει την δομή slice.
    - *struct tile* tl:* Δείκτης που περιγράφει την δομή tile.
    - *struct cu* cu:* Δείκτης που περιγράφει την δομή CU.
- **Struct slice:** Χρησιμοποιήθηκε για την αποθήκευση δεδομένων που προέκυψαν κατά την αποκωδικοποίηση και την οπτικοποίηση αυτών. Περιέχει δυο μεταβλητές ακέραιου τύπου.
    - *Int startCtu:* Ο αριθμός του CTU από το οποίο ξεκινάει το slice. Ο αριθμός είναι αύξων κατά οριζόντια σάρωση ανά σειρά.
    - *Int totalCTUofPrevSlices:* Ο συνολικός αριθμός των CTU που περιέχονται στα προηγούμενα slices.
- **Struct tile:** Έχει την ίδια χρήση με τα παραπάνω και περιέχει δύο μεταβλητές ακέραιου τύπου.
    - *Int firstCTUinTile:* Ο αριθμός του CTU από το οποίο ξεκινάει το tile. Ο αριθμός είναι αύξων κατά οριζόντια σάρωση ανά σειρά.
    - *Int totalCTUs:* Ο συνολικός αριθμός των CTU που περιλαμβάνονται στα προηγούμενα tiles.
- **Struct cu:** Είναι μία δομή δεδομένων που περιέχει μεταβλητές ακέραιου τύπου καθώς και μία μεταβλητή τύπου δεδομένων αληθείας, οι οποίες περιγράφουν το CU. Έχει την ίδια χρήση με τις παραπάνω δομές.
    - *Int CU_depth:* Το βάθος του δέντρου τετραδικής δομής στο οποίο βρίσκεται το CU.

- *Int CU_Left:* Η συντεταγμένη στον άξονα X που βρίσκεται το αριστερό σημείο του CU.
- *Int CU_Right:* Η συντεταγμένη στον άξονα X που βρίσκεται το δεξί σημείο του CU.
- *Int CU_Top:* Η συντεταγμένη στον άξονα Y που βρίσκεται το επάνω σημείο του CU.
- *Int CU_Bottom:* Η συντεταγμένη στον άξονα Y που βρίσκεται το κάτω σημείο του CU.
- *Int CU_Width:* Το πλάτος του βίντεο σε δείγματα luma.
- *Int CU_Height:* Το ύψος του βίντεο σε δείγματα luma.
- *Int CU_PredMode:* Ο τρόπος με τον οποίο κωδικοποιήθηκε το CU. Μπορεί να πάρει τρείς τιμές οι οποίες είναι 0 για την inter κωδικοποίηση, 1 για την intra και 2 εάν δεν κωδικοποιείται το CU.
- *Bool isMerged:* Boolean μεταβλητη που δείχνει εάν εχει γίνει merge

Ακολουθεί η δήλωση του namespace, το οποίο είναι μία συλλογή που περιλαμβάνει ορισμούς ονομάτων, ορισμούς κλάσεων και δηλώσεις μεταβλητών. Στην παρούσα υλοποίηση έχει χρησιμοποιηθεί το namespace HevcVAT. Εντός του namespace αυτού χρησιμοποιούνται και άλλα namespaces που παρέχονται μαζί με τον μεταγλωττιστή και αφορούν βασικές εντολές που υφίστανται στην C++. Το σημαντικότερο namespace που χρησιμοποιήθηκε είναι το System, καθώς περιέχει θεμελιώδεις κλάσεις που ορίζουν τύπους δεδομένων, events και τον χειρισμό αυτών καθώς και exceptions και τον χειρισμό αυτών. Στην συνέχεια ορίζεται η κλάση MainForm, η οποία είναι η σημαντικότερη κλάση της εφαρμογής καθώς εντός αυτής γίνεται ο χειρισμός των events που προκύπτουν από την αλληλεπίδραση με τον χρήστη και επίσης επιτυγχάνεται ο στόχος της εργασίας αυτής, η οπτικοποίηση των δομών που χρησιμοποιούνται για την κωδικοποίηση βίντεο, καθώς και η εμφάνιση στοιχείων και στατιστικών που αφορούν τις δομές αυτές. Η κλάση MainForm ξεκινάει με την δήλωση μεταβλητών και δομών που χρησιμοποιούνται από τις μεθόδους της καθώς και την δήλωση των ονομάτων των στοιχείων που υπάρχουν στο παράθυρο αυτό, δηλαδή κουμπιά, ετικέτες, πεδία κειμένου και πεδία αναπαραγωγής εικόνων. Ο κώδικας για την δήλωση των στοιχείων αυτών έχει δημιουργηθεί αυτόματα από το Microsoft Visual Studio κατά την διάρκεια ανάπτυξης του πρωτοτύπου στο περιβάλλον σχεδίασης. Παρακάτω θα παρουσιαστούν οι βασικότερες μέθοδοι της κλάσης MainForm και θα περιγραφούν λεπτομερώς.

- **CtuParser():** Είναι μία μέθοδος η οποία ανοίγει, διαβάζει ένα αρχείο κειμένου, το “ctu_info.txt” το οποίο προέκυψε από την αποκωδικοποίηση της ακολουθίας βίντεο και αποθηκεύει τα δεδομένα στην δομή frame. Το αρχείο αυτό περιέχει πληροφορίες όπως, ο αριθμός του τρέχοντος στιγμιότυπου, οι διαστάσεις του βίντεο σε pixel καθώς και σε CTUs. Κατά την διάρκεια της ανάγνωσης από το αρχείο γίνεται η απαραίτητη δέσμευση μνήμης στην δομή frame. Στο τέλος της ανάγνωσης από το αρχείο, το αρχείο διαγράφεται καθώς δεν είναι χρήσιμο πλέον.

- **SliceParser():** Είναι μία μέθοδος η οποία ανοίγει και διαβάζει ένα αρχείο κειμένου, το “slice_info.txt”. Είναι παρόμοια με την παραπάνω μέθοδο. Τα δεδομένα που ανακτώνται από το αρχείο αποθηκεύονται στην δομή Slice, η οποία είναι μέλος της δομής frame.
- **TileParser():** Είναι παρόμοια με τις παραπάνω μεθόδους. Ανοίγει ένα αρχείο με ονομασία “tile_info.txt” το οποίο προέκυψε κατά την διαδικασία της αποκωδικοποίησης επίσης. Τα δεδομένα που ανακτώνται από το αρχείο αποθηκεύονται στην δομή tile η οποία είναι μέλος της δομής frame.
- **CUParser():** Είναι παρεμφερής με τις παραπάνω μεθόδους. Ανοίγει ένα αρχείο κειμένου με όνομα “decodeCU_info.txt”. Το αρχείο περιέχει τις παρακάτω πληροφορίες για το κάθε frame του βίντεο, το βάθος του δέντρου που βρίσκεται το τρέχον CU, τον τρόπο με τον οποίο έγινε η πρόβλεψη για το τρέχον CU, το ύψος του, το πλάτος του καθώς και εάν έχει γίνει Merge ή όχι. Τα δεδομένα αυτά αποθηκεύονται στην δομή cu η οποία είναι μέλος της δομής frame.
- **StatsParser():** Είναι παρεμφερής με τις παραπάνω μεθόδους. Ανοίγει το αρχείο “cu_info.txt”, διαβάζει και αποθηκεύει δεδομένα όπως το τρέχον frame εντός του οποίου βρίσκεται το CU και τις συντεταγμένες των τεσσάρων άκρων του στην δομή cu, η οποία είναι μέλος της δομής frame.
- **SpsParser():** Είναι παρεμφερής με τις παραπάνω μεθόδους. Ανοίγει το αρχείο “decoder_sps.txt”. Η διαφορά με τις παραπάνω μεθόδους είναι ότι η SpsParser δεν αποθηκεύει δεδομένα σε κάποια δομή, αλλά χρησιμοποιείται για να εκτυπωθούν οι τιμές στο παράθυρο του χρήστη. Το αρχείο περιέχει πληροφορίες όπως: το πλάτος και το ύψος του βίντεο, το χρωματικό βάθος που έχει η ακολουθία, τον τύπο δειγματοληψίας που έχει χρησιμοποιηθεί, το μέγιστο επιτρεπτό βάθος που μπορεί να λάβει ένα CU, το μέγιστο επιτρεπτό μέγεθος που μπορεί να λάβει ένα CU καθώς και το μέγιστο βάθος της δομής τετραδικών δέντρων που μπορεί να λάβει ένα CU κωδικοποιημένο Inter είτε Intra.
- **cusToolStripMenuItem_Click():** Είναι μία μέθοδος η οποία διαχειρίζεται την αλληλεπίδραση του χρήστη με την εφαρμογή όταν επιλέγεται το κουμπί CU Structure. Όταν ενεργοποιείται η λειτουργία αυτή, μία δομή επανάληψης διαβάζει από την δομή frame τα περιεχόμενα όλων των CU και στην συνέχεια οπτικοποιεί τα δεδομένα αυτά και εμφανίζονται επί της εικόνας του βίντεο.
- **interToolStripMenuItem_Click():** Είναι μία μέθοδος η οποία διαχειρίζεται την αλληλεπίδραση του χρήστη με την εφαρμογή όταν επιλέγεται το κουμπί Inter. Η λειτουργεία της είναι παρόμοια με της παραπάνω μεθόδου. Μία δομή επανάληψης διαβάζει χαρακτηριστικά για το CU του τρέχοντος frame και τα οπτικοποιεί.
- **intraToolStripMenuItem_Click():** Είναι μία μέθοδος η οποία διαχειρίζεται την αλληλεπίδραση του χρήστη με την εφαρμογή όταν επιλέγεται το κουμπί Intra. Η λειτουργεία της είναι παρόμοια με της παραπάνω μεθόδου. Μία δομή

επανάληψης διαβάζει χαρακτηριστικά για το CU του τρέχοντος frame και τα οπτικοποιεί.

- **mergeToolStripMenuItem_Click():** Η λειτουργία της είναι ίδια με της παραπάνω. Διαχειρίζεται την αλληλεπίδραση του χρήστη με την εφαρμογή όταν επιλέγεται το κουμπί merge.

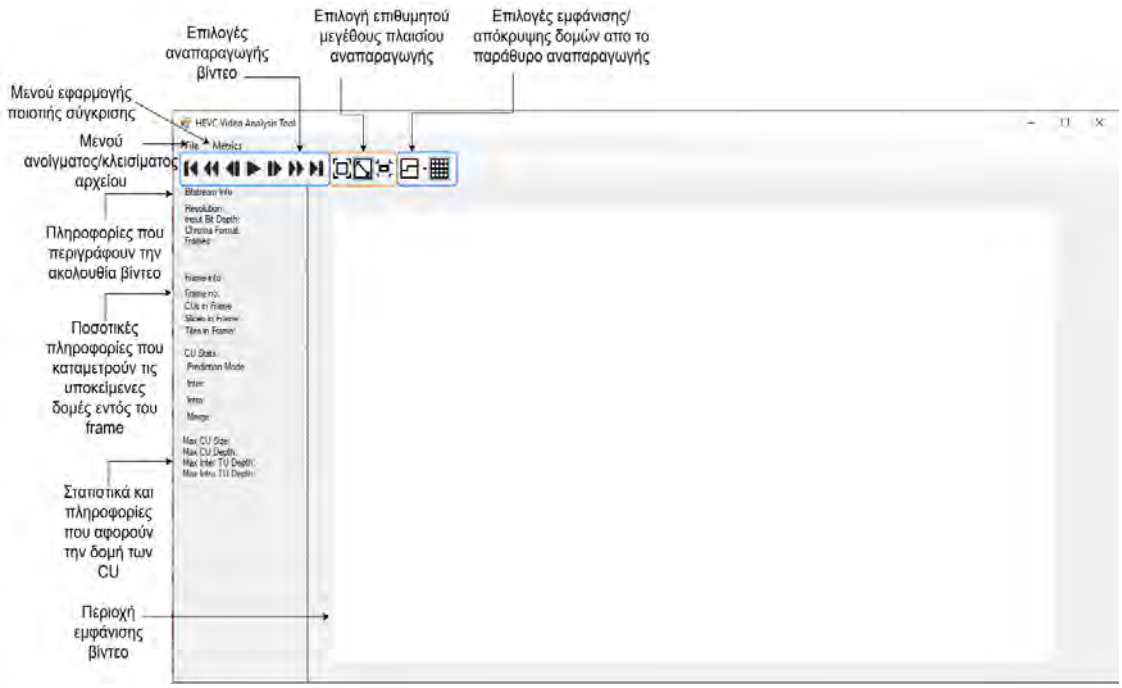
Στην συνέχεια, το αρχείο MainForm.cpp. Στο αρχείο αυτό βρίσκεται το σημείο εκκίνησης της εφαρμογής, η μέθοδος main. Εντός της main δημιουργείται ένα αντικείμενο της φόρμας MainForm και εκκινεί η εφαρμογή.

Το αρχείο PsnrForm.h αποτελεί ένα πλαίσιο που εμφανίζεται όταν επιλεχθεί η λειτουργεία PSNR. Είναι υπεύθυνο για την δήλωση του framerate και την μεταφορά του στο αρχείο Mainform.h.

Το αρχείο PSNR.h περιέχει την δήλωση των μεταβλητών και της μεθόδου που χρησιμοποιείται για τον υπολογισμό του PSNR, καθώς το αρχείο PSNR.cpp περιέχει την υλοποίηση της μεθόδου αυτής.

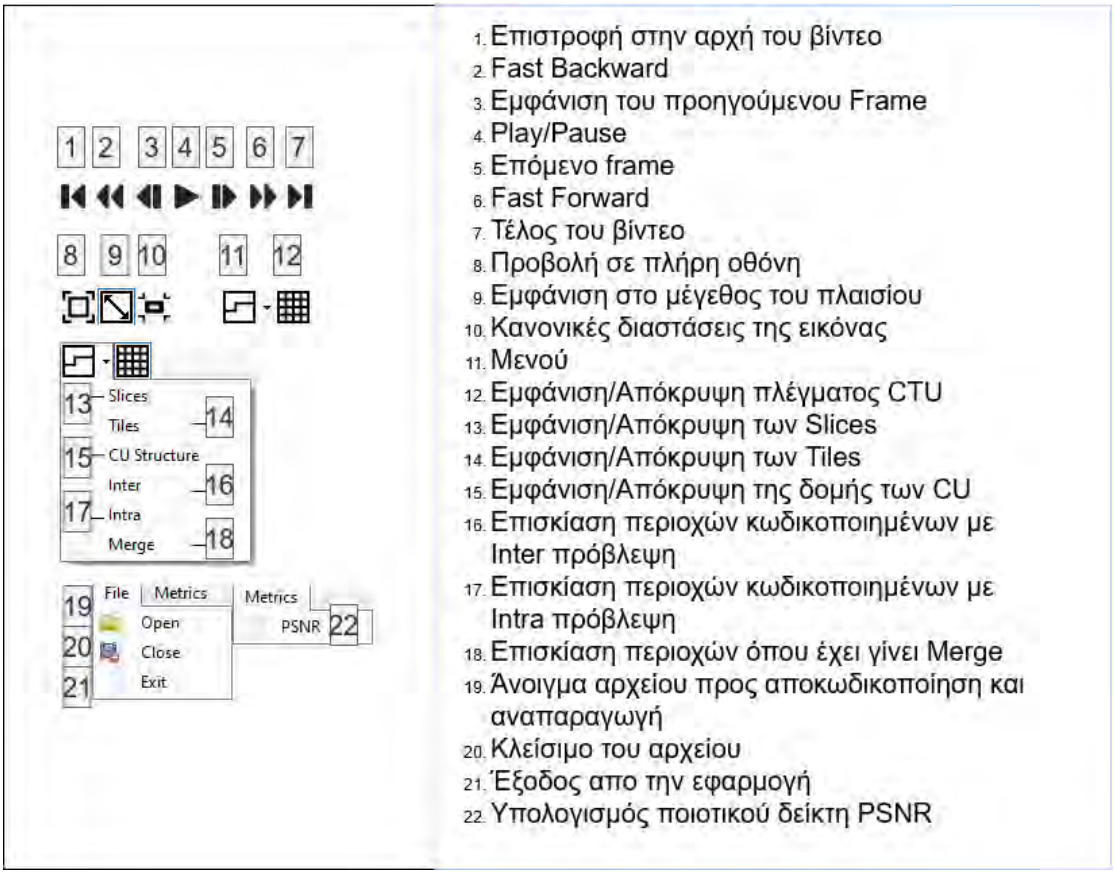
5.4. Παρουσίαση της Εφαρμογής HEVC Video Analysis Tool

Ακολουθεί η παρουσίαση της εφαρμογής. Όταν ο χρήστης εκκινήσει την εφαρμογή θα αντικρύσει ένα γραφικό περιβάλλον χρήστη το οποίο περιγράφεται από την Εικόνα 15.



Εικόνα 15: Το αρχικό παράθυρο του περιβάλλοντος της εφαρμογής

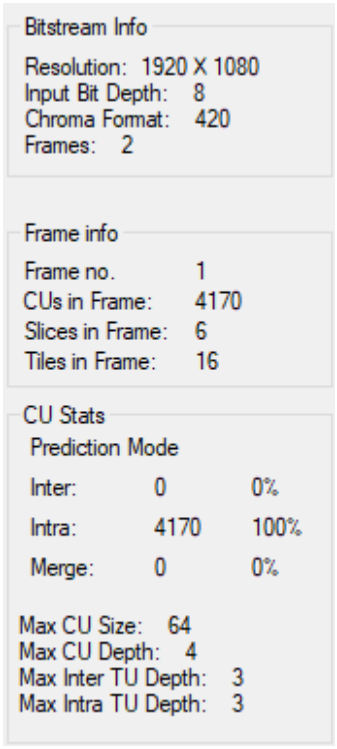
Η παρακάτω εικόνα (Εικόνα 16) συνοψίζει όλα τα κουμπιά που υπάρχουν στο παράθυρο της εφαρμογής και τις λειτουργείες τους. Μέσω αυτών των κουμπιών επιτυγχάνεται η αλληλεπίδραση του χρήστη με την εφαρμογή. Αρχικά ο χρήστης πρέπει να ανοίξει κωδικοποιημένη μια ακολουθία βίντεο (.bin), αυτό επιτυγχάνεται με την επιλογή "Open" που βρίσκεται εντός του μενού "File". Στην συνέχεια θα ανοίξει ένα παράθυρο πλοήγησης αρχείων, όπου ο χρήστης επιλέγει την ακολουθία ως προς αποκωδικοποίηση και αναπαραγωγή. Αμέσως μετά ανοίγει ένα δεύτερο παράθυρο πλοήγησης αρχείων όπου ο χρήστης καλείται να ορίσει ένα όνομα αρχείου ως προς αποθήκευση, έτσι ώστε να αποθηκευτεί το αποκωδικοποιημένο αρχείο. Αφού ολοκληρωθούν τα παραπάνω βήματα η εφαρμογή καλεί ένα ξεχωριστό εκτελέσιμο αρχείο, τον αποκωδικοποιητή, ο οποίος εκτελείται σε περιβάλλον γραμμής εντολών (cmd). Αφού ολοκληρωθεί η διαδικασία της αποκωδικοποίησης το παράθυρο της γραμμής εντολών κλείνει αυτόματα και πλέον στην εφαρμογή εμφανίζεται το πρώτο στιγμιότυπο του βίντεο.



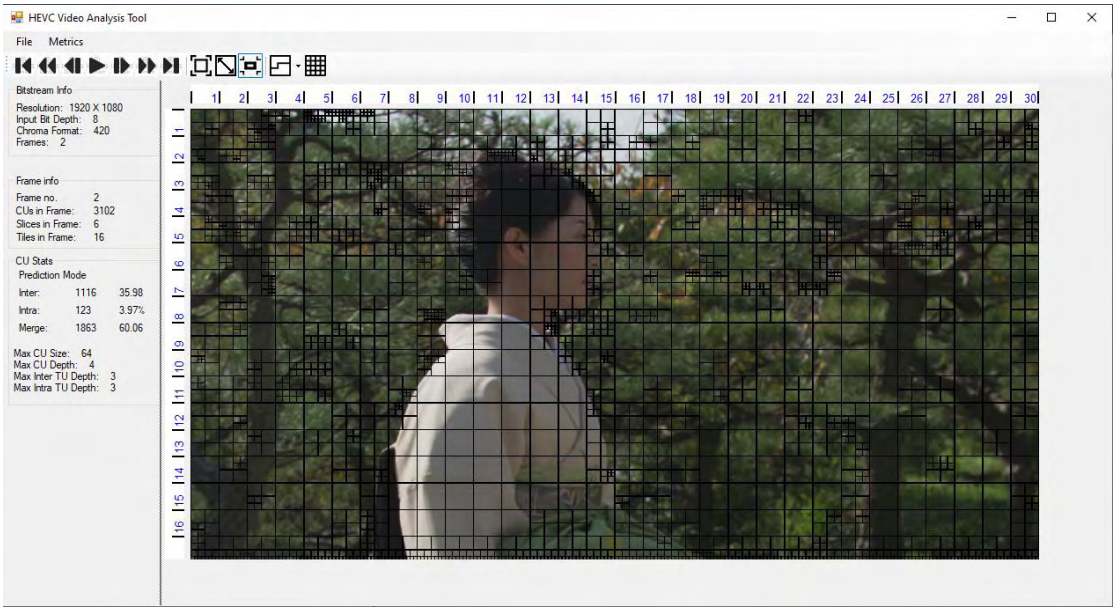
Εικόνα 16: Τα κουμπιά που βρίσκονται στο παράθυρο της εφαρμογής και η χρησιμότητά τους

Για τον έλεγχο της ορθότητας των αποτελεσμάτων χρησιμοποιήθηκε η ακολουθία " Kimono_1920x1080_24.yuv".

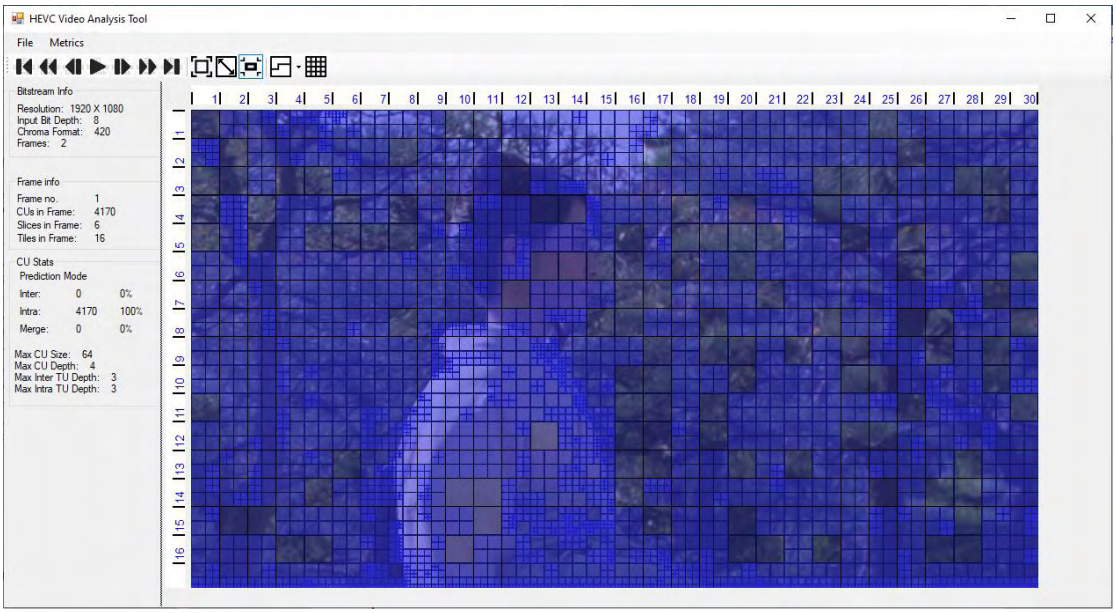
Εικόνα 17: Αφού φορτώθηκε η ακολουθία, η εφαρμογή αναπαράγει το πρώτο frame καθώς και τις πληροφορίες σχετικά με την ακολουθία καθώς και το frame



Εικόνα 18: Πληροφορίες που περιγράφουν την ακολουθία βίντεο, το τρέχον στιγμιότυπο καί τα περιεχόμενά του.

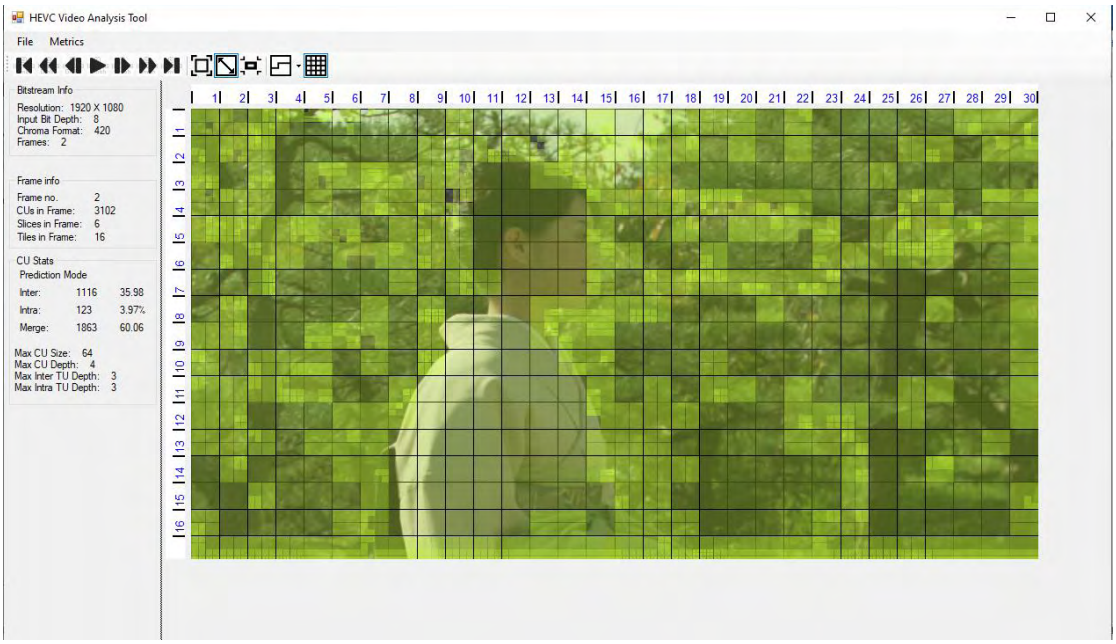


Εικόνα 19: Το δεύτερο frame της ακολουθίας με την επιλογή CU Structure ενεργοποιημένη

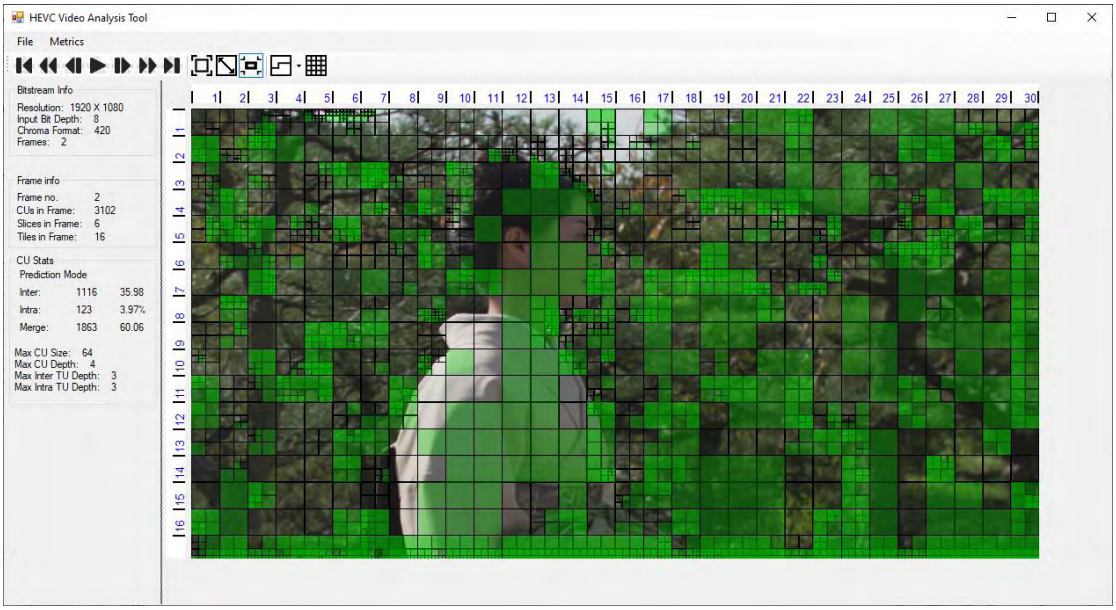


Εικόνα 20: Το πρώτο frame της ακολουθίας με τις επιλογές CU Structure και Intra ενεργοποιημένες

Εικόνα 21: Το 2ο frame της ακολουθίας με την επιλογή Intra ενεργοποιημένη

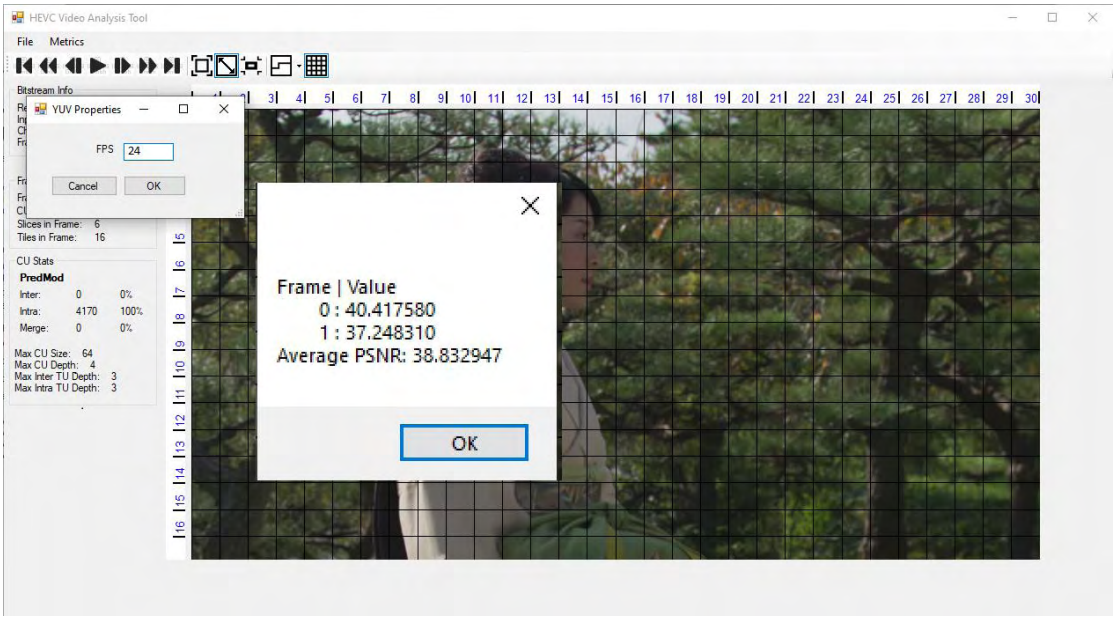


Εικόνα 22: Το 2ο frame της ακολουθίας με τις επιλογές Inter και CTU Grid ενεργοποιημένες



Εικόνα 23: Το 2ο frame της ακολουθίας με τις επιλογές CU Structure και Merge ενεργοποιημένες

Για τον υπολογισμό του PSNR αφού ο χρήστης επιλέξει το μενού “Metrics” και “PSNR” στην συνέχεια καλείται να ανοίξει το πρωτότυπο ασυμπίεστο αρχείο. Αφού επιλεγεί το αρχείο ο χρήστης καλείται να συμπληρώσει σε μια νέα φόρμα τον ρυθμό αναπαραγωγής των frame. Αφού ο χρήστης πατήσει “OK” η φόρμα θα κλείσει και θα εμφανιστεί ένα νέο παράθυρο που θα παρουσιάζει τις τιμές του PSNR για κάθε frame καθώς και τον μέσο όρο του PSNR όλων των frames, που περιγράφει την ακολουθία.



Εικόνα 24: Η φόρμα PsnrForm καθώς και το αποτέλεσμα του υπολογισμού του PSNR

6. Συμπεράσματα και Μελλοντικές Επεκτάσεις

Σκοπός της παρούσας διπλωματικής εργασίας ήταν η ανάπτυξη του εργαλείου ανάλυσης βίντεο "HEVC Video Analysis Tool" προσθέτοντας λειτουργείες που θα βοηθήσουν τον χρήστη να αντιληφθεί τις δομές που χρησιμοποιούνται από το HEVC καθώς και η παροχή πληροφοριών σχετικές με την ακολουθία βίντεο. Επίσης, η παροχή στατιστικών που αφορούν τον τρόπο πρόβλεψης σε συνδυασμό με τον δείκτη PSNR μπορεί να οδηγήσει τον χρήστη στην εξαγωγή συμπερασμάτων, καθώς μπορεί να κωδικοποιήσει την ίδια ακολουθία με διαφορετικές παραμέτρους και να συγκρίνει τα αποτελέσματα.

Για την υλοποίηση της εφαρμογής ήταν αναγκαίο το θεωρητικό υπόβαθρο στις τεχνικές κωδικοποίησης του HEVC καθώς και η γνώση των γλωσσών προγραμματισμού C & C++. Για την ανάπτυξη της εφαρμογής που περιέχει γραφική διεπαφή χρήστη χρησιμοποιήθηκε το Microsoft Visual Studio 19 Community Version. Για την συμπίεση, αποσυμπίεση και εξαγωγή των δεδομένων χρησιμοποιήθηκε το codec HEVC HM Reference Software [2].

Το HEVC Video Analysis Tool μπορεί να αναπτυχθεί περεταίρω. Προτάσεις για μελλοντικές επεκτάσεις είναι οι κάτωθι:

- Υλοποίηση της εφαρμογής με τέτοιο τρόπο ώστε να είναι cross-platform.
- Εξαγωγή δεδομένων και οπτικοποίηση των διανυσμάτων κίνησης (Motion Vectors)
- Υπολογισμός του δείκτη SSIM
- Λειτουργεία Split View έτσι ώστε ο χρήστης να μπορεί να συγκρίνει την ίδια ακολουθία βίντεο κωδικοποιημένη με διαφορετικές παραμέτρους.

Παράρτημα-Κώδικας

```
//HM Reference Software

//Project: Parser
//File: Ctu2Parser.h
//------------------
#ifndef __CTU2PARSER_H
#define __CTU2PARSER_H



#include "TLibCommon/TComDataCU.h"
#include "TLibCommon/TComSlice.h"
#include "TLibDecoder/TDecCu.h"
#include <fstream>


using namespace std;
class Ctu2Parser
{
public:

    Ctu2Parser() {
        m_cCTUOutput.open("cu_info.txt", ios::out);

    }
    ~Ctu2Parser()
    {
        m_cCTUOutput.close();
    }
    void Ctu2Parser::writeOutCtuInfo(TComDataCU* pCtu, UInt uiLPelX,
UInt uiRPelX, UInt uiTPelY, UInt uiBPelY,bool einaiSkipped, TComPic*
pcPic,UInt uiAbsPartIdx, UInt uiDepth)
    {
        m_cCTUOutput << "Frame: " << pcPic->getPOC() << "\n";
//number of frame
        m_cCTUOutput << "CU_depth: " << uiDepth << "\n";
        m_cCTUOutput << "predMode: " << pCtu-
>getPredictionMode(uiAbsPartIdx) << "\n";
        m_cCTUOutput << "Height: " << (int)pCtu-
>getHeight(uiAbsPartIdx) << "\n";
        m_cCTUOutput << "Width: " << (int)pCtu-
>getWidth(uiAbsPartIdx) << "\n\n";
        m_cCTUOutput << "isSkipped: " << einaiSkipped << "\n";
    }

    static Ctu2Parser* getInstance()
    {
        if (m_instance == NULL)
            m_instance = new Ctu2Parser;
        return m_instance;
    }
private:
    std::ofstream m_cCTUOutput;      ///< CU info output


    static Ctu2Parser* m_instance;
};
Ctu2Parser* Ctu2Parser::m_instance = NULL;
```

```
#endif // !__Ctu2Parser_H
#pragma once

//------------------

//File: CtuParser.h
//------------------
#ifndef __CTUPARSER_H
#define __CTUPARSER_H



#include "TLibCommon/TComDataCU.h"
#include "TLibCommon/TComSlice.h"
#include "TLibDecoder/TDecCu.h"
#include <fstream>


using namespace std;
class CtuParser
{
public:

    CtuParser() {
        m_cCTUOutput.open("decodeCU_info.txt", ios::out);

    }
    ~CtuParser()
    {
        m_cCTUOutput.close();
    }
    void CtuParser::writeOutCtuInfo(TComDataCU* pcCU, UInt uiLPelX,
UInt uiRPelX, UInt uiTPelY, UInt uiBPelY, const UInt uiDepth, UInt
uiCurNumParts, const UInt uiAbsPartIdx)
    {
        TComPic* pcPic = pcCU->getPic();

        ("decodeCU_info.txt", std::ofstream::app);
        m_cCTUOutput << "Frame: " << pcPic->getPOC() << "\n";
//number of frame
        m_cCTUOutput << "CU_depth: " << uiDepth << "\n";
        //m_cCTUOutput << "CTU_partitions no: " << uiCurNumParts <<
"\n";
        m_cCTUOutput << "CU-left-X: " << uiLPelX << "\n";
        m_cCTUOutput << "CU-right-X: " << uiRPelX << "\n";
        m_cCTUOutput << "CU-top-X: " << uiTPelY << "\n";
        m_cCTUOutput << "CU-bottom-X: " << uiBPelY << "\n\n";
        //m_cCTUOutput << "total_CTUs_in_frame: " << pcPic-
>getNumberOfCtusInFrame() << "\n\n";
    }

    static CtuParser* getInstance()
    {
        if (m_instance == NULL)
            m_instance = new CtuParser;
        return m_instance;
    }
private:
    std::ofstream m_cCTUOutput;      ///< CU info output
```

```
    static CtuParser* m_instance;
};
CtuParser* CtuParser::m_instance = NULL;

#endif // !__CtuParser_H
#pragma once

//------------------

//File: SliceParser.h
//------------------
#pragma once
#ifndef __SLICEPARSER_H
#define __SLICEPARSER_H



#include "TLibCommon/TComDataCU.h"
#include "TLibCommon/TComSlice.h"
#include "TLibCommon/TComTU.h"
#include "TLibDecoder/TDecCu.h"
#include <fstream>


using namespace std;
class SliceParser
{
public:

    SliceParser() {
        m_cSliceOutput.open("slice_info.txt", ios::out);
    }
    ~SliceParser()
    {
        m_cSliceOutput.close();
    }
    void SliceParser::writeOutSliceInfo(TComPic *pcPic, const Int
startCtuRsAddr,const Int startCtuTsAddr)
    {
        m_cSliceOutput << "Frame: " << pcPic->getPOC() << "\n";
//number of frame
        m_cSliceOutput << "StartCTU: " << startCtuRsAddr << "\n";
//start CTU of current slice
        m_cSliceOutput << "TotalCTUofPrevSlices: " << startCtuTsAddr
<< "\n\n";     //total CTUs of previous slices

    }

    static SliceParser* getInstance()
    {
        if (m_instance == NULL)
            m_instance = new SliceParser;
        return m_instance;
    }
private:
    std::ofstream m_cSliceOutput;        ///< Slice info output


    static SliceParser* m_instance;
};
```

```
SliceParser* SliceParser::m_instance = NULL;

#endif // !__SliceParser_H


//------------------

//File SPSParser.h
//------------------
#ifndef __SPSParser_H
#define __SPSParser_H



#include "TLibCommon/TComDataCU.h"
#include "TLibCommon/TComSlice.h"
#include "TLibDecoder/TDecCu.h"
#include <fstream>


using namespace std;

class SPSParser
{
public:

    SPSParser() {
        m_cSpsOut.open("decoder_sps.txt", ios::out);
    }
    ~SPSParser()
    {
        m_cSpsOut.close();
    }

    void SPSParser::writeSPS(TComSPS* pcSPS)
    {
        int iInputBitDepth = pcSPS->getBitDepth(CHANNEL_TYPE_LUMA);

        m_cSpsOut << "Width: " << pcSPS->getPicWidthInLumaSamples()
<< endl;
        m_cSpsOut << "Height: " << pcSPS->getPicHeightInLumaSamples()
<< endl;
        m_cSpsOut << "InputBitDepth: " << iInputBitDepth << endl;
        m_cSpsOut << "ChromaFormat: " << pcSPS->getChromaFormatIdc()
<< endl;
        m_cSpsOut << "MaxCUSize: " << pcSPS->getMaxCUHeight() <<
endl;
        m_cSpsOut << "MaxCUDepth: " << pcSPS->getMaxTotalCUDepth() <<
endl;
        m_cSpsOut << "MaxInterTUDepth: " << pcSPS-
>getQuadtreeTUMaxDepthInter() << endl;
        m_cSpsOut << "MaxIntraTUDepth: " << pcSPS-
>getQuadtreeTUMaxDepthIntra() << endl;
    }



    static SPSParser* getInstance()
    {
        if (m_instance == NULL)
            m_instance = new SPSParser;
```

```
        return m_instance;
    }

private:
    std::ofstream m_cSpsOut;          ///< SPS info

    static SPSParser* m_instance;
};
SPSParser* SPSParser::m_instance = NULL;


#endif // !__SPSParser_H

//------------------

//File: TileParser.h
//------------------
#pragma once
#ifndef __TILEPARSER_H
#define __TILEPARSER_H



#include "TLibCommon/TComDataCU.h"
#include "TLibCommon/TComSlice.h"
#include "TLibCommon/TComTU.h"
#include "TLibDecoder/TDecCu.h"
#include <fstream>


using namespace std;
class TileParser
{
public:

    TileParser() {
        m_cTileOutput.open("tile_info.txt", ios::out);
    }
    ~TileParser()
    {
        m_cTileOutput.close();
    }
    void TileParser::writeOutTileInfo(TComPic* pcPic, int
totalCTUinTile, TComTile currentTile)
    {
        m_cTileOutput << "Frame: " << pcPic->getPOC() << "\n";
//number of frame
        m_cTileOutput << "totalCTUinTile: " << totalCTUinTile <<
"\n";                              //total CTUs in current tile
        m_cTileOutput << "ctuRsAddr: " <<
currentTile.getFirstCtuRsAddr() << "\n\n";                //first
tile in current tile
    }

    static TileParser* getInstance()
    {
        if (m_instance == NULL)
            m_instance = new TileParser;
        return m_instance;
    }
private:
```

```
    std::ofstream m_cTileOutput;        ///< TU info output


    static TileParser* m_instance;
};
TileParser* TileParser::m_instance = NULL;

#endif // !__TileParser_H


//------------------

//Project: TLibDecoder
//File: TDecCU.cpp
//Function: xDecompressCU
//Lines: 379-380
  bool isSkipped = pCtu->getSkipFlag(uiAbsPartIdx) ? 1 : 0;
  Ctu2Parser::getInstance()->writeOutCtuInfo(pCtu, uiLPelX, uiRPelX,
uiTPelY, uiBPelY,isSkipped,pcPic, uiAbsPartIdx,uiDepth);
//------------------

//File: TDecCU.cpp
//Function: xDecodeCU
//Line: 220
  CtuParser::getInstance()->writeOutCtuInfo(pcCU, uiLPelX, uiRPelX,
uiTPelY, uiBPelY, uiDepth, uiCurNumParts, uiAbsPartIdx);
//------------------

//File: TDecSlice.cpp
//Function: decompressSlice
//Line: 95
  SliceParser::getInstance()->writeOutSliceInfo(pcPic,
startCtuRsAddr, startCtuTsAddr);
//------------------

//File: TDecSlice.cpp
//Function: decompressSlice
//Line: 186
  TileParser::getInstance()->writeOutTileInfo(pcPic, totalCTUinTile,
currentTile);
//------------------

//File: TDecTop.cpp
//Function: xDecodeSPS
//Line: 732
  SPSParser::getInstance()->writeSPS(sps);
//------------------

//File: TDecSlice.cpp
//Function: decompressSlice
//Line: 186
  TileParser::getInstance()->writeOutTileInfo(pcPic, totalCTUinTile,
currentTile);
//------------------

//**************************************************************************
**************
// HEVC Video Analysis Tool
//**************************************************************************
**************
//File: MainForm.h
```

```
//Lines: 69-70
//Added two struct pointers in the struct Frame
  struct cu *cu;
  struct tu *tu;
//------------------

//File: MainForm.h
//Lines: 90-101
//New struct cu that carries info for CUs
  struct cu
{
    int CU_depth;
    int CU_Left;
    int CU_Right;
    int CU_Top;
    int CU_Bottom;
    int CU_Height;
    int CU_Width;
    int CU_PredMode;
    bool isSkipped;
};
//------------------

//File: MainForm.h
//Line: 110
//Declared an yuv capture that carries the raw video in order to
compute the psnr
struct YUV_Capture raw_cap;
//------------------

//File: MainForm.h
//Line: 145
//declaration of raw video file
FILE *raw_fin = NULL;
//------------------

//File: MainForm.h
//Line: 153-156
//Added declaration of variables inside MainForm Function
char* proccessed_path;
int encWidth;
int encHeight;
int chromaFormat;
//------------------

//File: MainForm.h
//Line: 339-340
//Called the functions inside ShowMyImage Function
CtuParser();
SpsParser();
//------------------

//File: MainForm.h
//Line: 349
//Allaksa thn grammh auth etsi wste na anoigei video pera apo
1920x1080 pou htan sthn arxh, bgr = cvCreateImage(cvSize(1920, 1080),
IPL_DEPTH_8U, 3);
bgr = cvCreateImage(cvSize(encWidth, encHeight), IPL_DEPTH_8U, 3);
//------------------

//File: MainForm.h
```

```
//Line: 360-361
//Called the functions
CUParser();
StatsParser();
//------------------

//File: MainForm.h
//Function: CUParser
//Line: 614-673
void CUParser() {
    std::ifstream infile("decodeCU_info.txt");
    std::string line;
    std::string number;

    int num, l = 1, fr = 0, CUs = 0;                    //num carries
the integer number of current line
                                                        //l defines
the current line
                                                        //fr stores
the current frame

    while (std::getline(infile, line, ' '))
    {

        std::getline(infile, number, '\n');     //store the string
number of a line
        num = std::stoi(number);                //convert string
number of a line into integer number

        switch (l)
        {
        case 1:
            if (fr != num) {
                fr++;
                CUs = 0;
            }
            break;
        case 2:
            if (CUs == 0) {
                pics[fr].cu = (struct cu*) malloc(sizeof(struct cu));
            }
            else {
                pics[fr].cu = (struct cu*) realloc(pics[fr].cu, (CUs
+ 1) * sizeof(struct cu));
            }
            pics[fr].cu[CUs].CU_depth = num;
            break;
        case 3:
            pics[fr].cu[CUs].CU_Left = num;
            break;
        case 4:
            pics[fr].cu[CUs].CU_Right = num;
            break;
        case 5:
            pics[fr].cu[CUs].CU_Top = num;
            break;
        case 6:
            pics[fr].cu[CUs].CU_Bottom = num;
            break;
        }
```

```
        l++;
        if (l == 7) {
            CUs += 1;
            pics[fr].numOfCUs = CUs;
            l = 1;
            }
        }
        infile.close();
        remove("decodeCU_info.txt");
}
//------------------

//File: MainForm.h
//Function: StatsParser
//Line: 675-724
void StatsParser() {
    std::ifstream infile("cu_info.txt");
    std::string line;
    std::string number;
    int num, l = 1, fr = 0, CUs = 0;

    while (std::getline(infile, line, ' '))
    {

        std::getline(infile, number, '\n');     //store the string
number of a line
        num = std::stoi(number);                //convert string
number of a line into integer number

        switch (l)
        {
        case 1:
            if (fr != num) {
                fr++;
                CUs = 0;
            }
            break;
        case 2:
            break;
        case 3:
            pics[fr].cu[CUs].CU_PredMode = num;
            break;
        case 4:
            pics[fr].cu[CUs].CU_Height = num;
            break;
        case 5:
            pics[fr].cu[CUs].CU_Width = num;
            break;
        case 6:
            pics[fr].cu[CUs].isSkipped = num;
            break;
        }

        l++;
        if (l == 7) {
            l = 1;
            CUs += 1;
        }

    }
    infile.close();
```

```
    remove("cu_info.txt");
//------------------

//File: MainForm.h
//Function: SpsParser
//Line: 779-847
void SpsParser() {
    std::ifstream infile("decoder_sps.txt");
    std::string line;
    std::string number;

    int num, l = 1;

    while (std::getline(infile, line, ' '))
    {

        std::getline(infile, number, '\n');     //store the string
number of a line
        num = std::stoi(number);                //convert string
number of a line into integer number
        String^ widthLbString;
        String^ heightLbString;
        String^ chromaFormatStr;
        int inputBitDepth, chromaFormat, maxCUSize, maxCUDepth,
maxInterTUDepth, maxIntraTUDepth;
        switch (l)
        {
        case 1:
            encWidth = num;
            widthLbString = encWidth.ToString();
            break;
        case 2:
            encHeight = num;
            heightLbString = encHeight.ToString();
            resolutionLabel->Text = encWidth.ToString() + " X " +
encHeight.ToString();
            break;
        case 3:
            inputBitDepth = num;
            inputBitDepthLabel->Text = inputBitDepth.ToString();
            break;
        case 4:
            chromaFormat = num;
            chromaFormatStr = gcnew String(number.c_str());

            if (chromaFormat == 0) chromaFormatStr = "440";
            else if (chromaFormat == 1) chromaFormatStr = "420";
            else if (chromaFormat == 2) chromaFormatStr = "422";
            else if (chromaFormat == 3) chromaFormatStr = "444";
            chromaFormatLabel->Text = chromaFormatStr;
            break;
        case 5:
            maxCUSize = num;
            maxCUSizeLabel->Text = maxCUSize.ToString();
            break;
        case 6:
            maxCUDepth = num;
            maxCUDepthLabel->Text = maxCUDepth.ToString();
            break;
        case 7:
            maxInterTUDepth = num;
```

```
            maxInterTUDepthLabel->Text = maxInterTUDepth.ToString();
            break;
        case 8:
            maxIntraTUDepth = num;
            maxIntraTUDepthLabel->Text = maxIntraTUDepth.ToString();
            break;
        }

        l++;
        if (l == 9) l = 1;
    }
    infile.close();
    remove("decoder_sps.txt");
}
//------------------

//File: MainForm.h
//Function: PictureBox1_Grid
//Line: 966-988
float CULPos, CURPos, CUTPos, CUBPos;
//if the relevant menu button is clicked
if (cusToolStripMenuItem->Checked)
{
    float x1=0, x2=0, y1=0, y2=0;
    myPen = gcnew Pen(Color::Black, 1);
    for (int i = 0; i < pics[cur_frame].numOfCUs; i++)
    {
        CULPos = pics[cur_frame].cu[i].CU_Left;
        CURPos = pics[cur_frame].cu[i].CU_Right;
        CUTPos = pics[cur_frame].cu[i].CU_Top;
        CUBPos = pics[cur_frame].cu[i].CU_Bottom;

        x1 = CULPos / pics[cur_frame].frameWidth * picWidth;
        x2 = CURPos / pics[cur_frame].frameWidth * picWidth;

        y1 = CUTPos / pics[cur_frame].frameHeight * picHeight;
        y2 = CUBPos / pics[cur_frame].frameHeight * picHeight;

        gr->DrawRectangle(myPen, x1, y1, x2 - x1, y2 - y1);
    }

}
//------------------

//File: MainForm.h
//Function: PictureBox1_Grid
//Line: 1023-1086
if (interToolStripMenuItem->Checked)
{
    float x1 = 0, x2 = 0, y1 = 0, y2 = 0;
    Color brushColor = Color::FromArgb(75, 204, 255, 51);
    SolidBrush^ myBrush = gcnew SolidBrush(brushColor);
    for (int i = 0; i < pics[cur_frame].numOfCUs; i++)
    {
        CULPos = pics[cur_frame].cu[i].CU_Left;
        CURPos = pics[cur_frame].cu[i].CU_Right;
        CUTPos = pics[cur_frame].cu[i].CU_Top;
        CUBPos = pics[cur_frame].cu[i].CU_Bottom;

        x1 = CULPos / pics[cur_frame].frameWidth * picWidth;
        x2 = CURPos / pics[cur_frame].frameWidth * picWidth;
```

```
            y1 = CUTPos / pics[cur_frame].frameHeight * picHeight;
            y2 = CUBPos / pics[cur_frame].frameHeight * picHeight;
            if (pics[cur_frame].cu[i].CU_PredMode == 0)
            gr->FillRectangle(myBrush, x1, y1, x2 - x1, y2 - y1);
        }
    }
    if (intraToolStripMenuItem->Checked)
    {

        float x1 = 0, x2 = 0, y1 = 0, y2 = 0;
        Color brushColor = Color::FromArgb(75, 51, 51, 255);
        SolidBrush^ myBrush = gcnew SolidBrush(brushColor);
        for (int i = 0; i < pics[cur_frame].numOfCUs; i++)
        {
            CULPos = pics[cur_frame].cu[i].CU_Left;
            CURPos = pics[cur_frame].cu[i].CU_Right;
            CUTPos = pics[cur_frame].cu[i].CU_Top;
            CUBPos = pics[cur_frame].cu[i].CU_Bottom;

            x1 = CULPos / pics[cur_frame].frameWidth * picWidth;
            x2 = CURPos / pics[cur_frame].frameWidth * picWidth;

            y1 = CUTPos / pics[cur_frame].frameHeight * picHeight;
            y2 = CUBPos / pics[cur_frame].frameHeight * picHeight;
            if (pics[cur_frame].cu[i].CU_PredMode == 1)
                gr->FillRectangle(myBrush, x1, y1, x2 - x1, y2 - y1);
        }
    }
    if (mergeToolStripMenuItem->Checked)
    {
        float x1 = 0, x2 = 0, y1 = 0, y2 = 0;
        Color brushColor = Color::FromArgb(80, 0, 204, 0);
        SolidBrush^ myBrush = gcnew SolidBrush(brushColor);
        for (int i = 0; i < pics[cur_frame].numOfCUs; i++)
        {
            CULPos = pics[cur_frame].cu[i].CU_Left;
            CURPos = pics[cur_frame].cu[i].CU_Right;
            CUTPos = pics[cur_frame].cu[i].CU_Top;
            CUBPos = pics[cur_frame].cu[i].CU_Bottom;

            x1 = CULPos / pics[cur_frame].frameWidth * picWidth;
            x2 = CURPos / pics[cur_frame].frameWidth * picWidth;

            y1 = CUTPos / pics[cur_frame].frameHeight * picHeight;
            y2 = CUBPos / pics[cur_frame].frameHeight * picHeight;
            if (pics[cur_frame].cu[i].isSkipped == 1)
                gr->FillRectangle(myBrush, x1, y1, x2 - x1, y2 - y1);
        }
    }
    //------------------
    //File: MainForm.h
    //Function: PictureBox1_Grid
    //Line: 1089-1118
    label1->Text = (cur_frame+1).ToString();
    label6->Text = pics[cur_frame].numOfCUs.ToString();
    sliceNoLabel->Text = pics[cur_frame].numOfSlices.ToString();
    FramesCntLabel->Text = total_frames.ToString();
    tileNoLabel->Text = pics[cur_frame].numOfTiles.ToString();
    float interCntr = 0, intraCntr = 0,skipCntr =
    0,interPrcnt,intraPrcnt,skipPrcnt;
```

```
for (int i = 0; i < pics[cur_frame].numOfCUs; i++)
{
    if (pics[cur_frame].cu[i].CU_PredMode == 0)
        interCntr += 1;
    if (pics[cur_frame].cu[i].CU_PredMode == 1)
        intraCntr += 1;
    if (pics[cur_frame].cu[i].isSkipped == 1)
    {
        skipCntr  += 1;
        interCntr -= 1;
    }
}
interPrcnt =  round((interCntr /
pics[cur_frame].numOfCUs)*100.0*100.0)/100.0;
intraPrcnt =  round((intraCntr /
pics[cur_frame].numOfCUs)*100.0*100.0)/100.0;
skipPrcnt  =  round((skipCntr  /
pics[cur_frame].numOfCUs)*100.0*100.0)/100.0;


labelInterPercent->Text = interPrcnt.ToString()+"%";
labelIntraPercent->Text = intraPrcnt.ToString() + "%";
labelSkipPercent->Text  = skipPrcnt.ToString() + "%";

labelIntraCnt->Text = intraCntr.ToString();
labelInterCnt->Text = interCntr.ToString();
labelSkipCnt->Text  = skipCntr.ToString();
//------------------

//File: MainForm.h
//Function: pSNRToolStripMenuItem_Click
//Line: 2176-2291
private: System::Void pSNRToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e)
{

    Stream^ myStream;
    //initialize the objects by calling the constructor
    OpenFileDialog^ openFileDialog1 = gcnew OpenFileDialog;
    //define properties to the dialog objects
    openFileDialog1->InitialDirectory = "C:\\";
    openFileDialog1->Filter = "yuv files (*.yuv)|*.yuv|All files
(*.*)|*.*";
    openFileDialog1->FilterIndex = 1;
    openFileDialog1->RestoreDirectory = true;
    //if ok button is pressed and file isn't null, proceed
    if (openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK)
    {

        if ((myStream = openFileDialog1->OpenFile()) != nullptr)
        {
            //check if selected file has extension .yuv
            if (!openFileDialog1->FileName->EndsWith(".yuv"))
            {
                System::Windows::Forms::MessageBox::Show(
                    "Error opening file!!\nFilename extension must be
.yuv",
                    "File Warning",
                    MessageBoxButtons::OK,
                    MessageBoxIcon::Warning
```

```
                );
                return;
            }
            PsnrForm^ f1 = gcnew PsnrForm();
            f1->ShowDialog();

            String^ managed_fps = f1->getFps();
            std::string raw_fps =
msclr::interop::marshal_as<std::string>(managed_fps);
            std::string::size_type sz;
            std::string FileName =
msclr::interop::marshal_as<std::string>(openFileDialog1->FileName);
            const char* chr = FileName.c_str();         //raw file
path

            int iRaw_width = encWidth;
            int iRaw_height = encHeight;
            int iRaw_fps = std::stoi(raw_fps, &sz);
            int iRaw_chroma = chromaFormat;
            if (iRaw_chroma == 1) iRaw_chroma = 420;


            int nbframes = total_frames;
            VideoYUV* original = new VideoYUV(chr, iRaw_height,
iRaw_width, nbframes, iRaw_chroma);
            VideoYUV* processed = new VideoYUV(proccessed_path,
iRaw_height, iRaw_width, nbframes, iRaw_chroma);


            FILE* result_file;
            result_file = fopen("psnr.txt", "w");
            fprintf(result_file, "Frame | Value\n");
            //outfile1 << "frame,value\n";
            PSNR* psnr = new PSNR(iRaw_height, iRaw_width);

            cv::Mat original_frame(iRaw_height, iRaw_width, CV_32F),
processed_frame(iRaw_height, iRaw_width, CV_32F);
            float result;
            float result_avg;

            for (int frame = 0; frame < nbframes; frame++) {
                // Grab frame
                if (!original->readOneFrame()) exit(EXIT_FAILURE);
                original->getLuma(original_frame, CV_32F);
                if (!processed->readOneFrame()) exit(EXIT_FAILURE);
                processed->getLuma(processed_frame, CV_32F);

////////////////////////////////////////////////////////////////////
/////
                /*original = YUV_read(&raw_cap);
                processed= YUV_read(&prc_cap);

                if ((original == YUV_EOF)|| (processed == YUV_EOF))
                {
                    cvWaitKey(0);
                    return;
                }
                else if ((original == YUV_IO_ERROR)|| (processed ==
YUV_IO_ERROR))
                {
                    fprintf(stderr, "I/O error\n");
```

```
                    return;
                }
                raw_cap.YUV_getLuma(original_frame, CV_32F);
                prc_cap.YUV_getLuma(processed_frame, CV_32F);*/

////////////////////////////////////////////////////////////////////
/////
                // Compute PSNR
                if (result_file != NULL) {
                    //result = psnr->compute(original_frame,
processed_frame);
                    result = psnr->compute(original_frame,
processed_frame);
                }

                if (result_file != NULL) {
                    result_avg += result;
                    fprintf(result_file, "        %d : %.6f\n",
frame, static_cast<double>(result));
                    //outfile1 <<frame << " " <<result <<"\n";
                }

            }
            if (result_file != NULL) {
                result_avg /= static_cast<float>(nbframes);
                fprintf(result_file, "Average PSNR: %.6f",
static_cast<double>(result_avg));
                //outfile1 << "average: " << result_avg << "\n";
                fclose(result_file);
                //outfile1.close();
            }
            String^ filename = "psnr.txt";
            String^ ReadFile = File::ReadAllText(filename);
            MessageBox::Show(ReadFile);
            myStream->Close();
            remove("psnr.txt");                 //delete the text
file
            //remove("params.txt");                 //delete the text
file

            delete psnr;
            //delete original;
            //delete processed;

        }
    }
}
//------------------
//File: MainForm.h
//Function: toolStripButton11_Click
//Line: 2641-2645
//fixed the location and size of picture when normal image mode is
enabled
pictureBox1->SizeMode = PictureBoxSizeMode::StretchImage;

pictureBox1->Location = Point(32, 33);
pictureBox2->Location = Point(3, 33);
pictureBox3->Location = Point(32, 5);

pictureBox1->Size = System::Drawing::Size(964, 512);
//------------------
```

```
//File: MainForm.h
//Function: toolStripButton10_Click
//Line: 2669-2676
//fixed the location and size of picture when streched image mode is
enabled
    pictureBox1->Location = Point(32, 33);
    pictureBox2->Location = Point(3, 33);
    pictureBox3->Location = Point(32, 5);
    this->ClientSize = System::Drawing::Size(1260, 653);

    int pictWidth = this->splitContainer1->Panel2->Width;
    int pictHeight = this->splitContainer1->Panel2->Height;
    pictureBox1->Size = System::Drawing::Size(pictWidth-32,
pictHeight-33);
//------------------

//File: MainForm.h
//Function: toolStripButton10_Click
//Line: 2669-2676
//fixed the location and size of picture when fullscreen image mode
is enabled
    pictureBox1->Location = Point(32, 33);
    pictureBox2->Location = Point(3, 33);
    pictureBox3->Location = Point(32, 5);
//------------------

//File: MainForm.h
//Function: closeToolStripMenuItem_Click
//Line: 2845-2842
//when video closes labels displaying info reset to " ".
this->resolutionLabel->Text = "";
this->inputBitDepthLabel->Text = "";
this->chromaFormatLabel->Text = " ";
this->FramesCntLabel->Text = " ";
this->label1->Text = " ";
this->label6->Text = " ";
this->sliceNoLabel->Text = " ";
this->tileNoLabel->Text = " ";
this->labelInterCnt->Text = " ";
this->labelInterPercent->Text = " ";
this->labelIntraCnt->Text = " ";
this->labelIntraPercent->Text = " ";
this->labelSkipCnt->Text = " ";
this->labelSkipPercent->Text = " ";
this->maxCUSizeLabel->Text = " ";
this->maxCUDepthLabel->Text = " ";
this->maxInterTUDepthLabel->Text = " ";
this->maxIntraTUDepthLabel->Text = " ";
//------------------

//File: MainForm.h
//Function: cusToolStripMenuItem_Click
//Line: 2867-2884
//cu structure button event handling function
private: System::Void cusToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^  e) {
    if (cusToolStripMenuItem->Checked) {
        cusToolStripMenuItem->Checked = false;
        toolStripButton12->Checked=false;
    }
```

```
    else {
        cusToolStripMenuItem->Checked = true;
        toolStripButton12->Checked = false;

    }

    if (pics == nullptr) return;

        cvCvtColor(cap.ycrcb, bgr, CV_YCrCb2BGR);
        DrawMyImage(bgr);

        pictureBox1->Refresh();
}
//------------------

//File: MainForm.h
//Function: interToolStripMenuItem_Click
//Line: 2927-2941
//inter button event handling function
private: System::Void interToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
    if (interToolStripMenuItem->Checked) {
        interToolStripMenuItem->Checked = false;
    }
    else {
        interToolStripMenuItem->Checked = true;
    }

    if (pics == nullptr) return;

    cvCvtColor(cap.ycrcb, bgr, CV_YCrCb2BGR);
    DrawMyImage(bgr);

    pictureBox1->Refresh();
}
//------------------

//File: MainForm.h
//Function: intraToolStripMenuItem_Click
//Line: 2942-2956
//intra button event handling function
private: System::Void intraToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
    if (intraToolStripMenuItem->Checked) {
        intraToolStripMenuItem->Checked = false;
    }
    else {
        intraToolStripMenuItem->Checked = true;
    }

    if (pics == nullptr) return;

    cvCvtColor(cap.ycrcb, bgr, CV_YCrCb2BGR);
    DrawMyImage(bgr);

    pictureBox1->Refresh();
}
//------------------

//File: MainForm.h
//Function: mergeToolStripMenuItem_Click
```

```
//Line: 2958-2972
//merge button event handling function
private: System::Void mergeToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e) {
    if (mergeToolStripMenuItem->Checked) {
        mergeToolStripMenuItem->Checked = false;
    }
    else {
        mergeToolStripMenuItem->Checked = true;
    }

    if (pics == nullptr) return;

    cvCvtColor(cap.ycrcb, bgr, CV_YCrCb2BGR);
    DrawMyImage(bgr);

    pictureBox1->Refresh();
}
//------------------

//File: MainForm.h
//Function: splitContainer1_SplitterMoved
//Line: 3009-3026
//split container move button event handling function (resize
piturebox,groupboxes)
private: System::Void splitContainer1_SplitterMoved(System::Object^
sender, System::Windows::Forms::SplitterEventArgs^ e) {
    int pictWidth = this->splitContainer1->Panel2->Width;
    int pictHeight = this->splitContainer1->Panel2->Height;
    pictureBox1->Size = System::Drawing::Size(pictWidth-32,
pictHeight-33);
    pictureBox2->Height = pictureBox1->Height;
    pictureBox3->Width = pictureBox1->Width;

    pictureBox1->Refresh();
    pictureBox2->Refresh();
    pictureBox3->Refresh();
    int gbxWidth = this->splitContainer1->Panel1->Width;
    groupBox1->Size = System::Drawing::Size(gbxWidth-3, 85);
    groupBox2->Size = System::Drawing::Size(gbxWidth - 3, 83);
    groupBox3->Size = System::Drawing::Size(gbxWidth-3, 172);

    groupBox1->Refresh();
    groupBox3->Refresh();
}
//------------------

//File: PSNR.h
#include <cmath>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>

class PSNR {
public:
    int height;
    int width;
    PSNR(int height, int width);
    // Compute the PSNR index of the processed image
    float compute(const cv::Mat& original, const cv::Mat& processed);

};
```

```
//------------------

//File: PSNR.cpp
#include "PSNR.h"

PSNR::PSNR(int h, int w)
{
    height = h;
    width = w;
}

float PSNR::compute(const cv::Mat& original, const cv::Mat&
processed)
{
    cv::Mat tmp(height, width, CV_32F);
    cv::subtract(original, processed, tmp);
    cv::multiply(tmp, tmp, tmp);
    return float(10 * log10(255 * 255 / cv::mean(tmp).val[0]));
}
//------------------

//File: PsnrForm.h
#pragma once
#include <string.h>
#include <msclr\marshal.h>
#include <msclr\marshal_cppstd.h>
namespace Project3 {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for PsnrForm
    /// </summary>
    public ref class PsnrForm : public System::Windows::Forms::Form
    {
        String^ managed_fps;
    private: System::Windows::Forms::TextBox^ FPStextBox;
    private: System::Windows::Forms::Label^ label2;

    public: String^ getFps() { return managed_fps; }

    public:
        PsnrForm(void)
        {
            InitializeComponent();
        }

    protected:

        ~PsnrForm()
        {
            if (components)
            {
                delete components;
            }
        }
```

```
    protected:
    protected:

    private: System::Windows::Forms::Button^ button1;
    private: System::Windows::Forms::Button^ CancelButton;

    protected:

    private:
        System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void)
        {
            this->button1 = (gcnew System::Windows::Forms::Button());
            this->CancelButton = (gcnew
System::Windows::Forms::Button());
            this->FPStextBox = (gcnew
System::Windows::Forms::TextBox());
            this->label2 = (gcnew System::Windows::Forms::Label());
            this->SuspendLayout();
            //
            // button1
            //
            this->button1->Location = System::Drawing::Point(110,
60);
            this->button1->Name = L"button1";
            this->button1->Size = System::Drawing::Size(71, 23);
            this->button1->TabIndex = 9;
            this->button1->Text = L"OK";
            this->button1->UseVisualStyleBackColor = true;
            this->button1->Click += gcnew System::EventHandler(this,
&PsnrForm::button1_Click);
            //
            // CancelButton
            //
            this->CancelButton->Location = System::Drawing::Point(28,
60);
            this->CancelButton->Name = L"CancelButton";
            this->CancelButton->Size = System::Drawing::Size(75, 23);
            this->CancelButton->TabIndex = 10;
            this->CancelButton->Text = L"Cancel";
            this->CancelButton->UseVisualStyleBackColor = true;
            this->CancelButton->Click += gcnew
System::EventHandler(this, &PsnrForm::CancelButton_Click);
            //
            // FPStextBox
            //
            this->FPStextBox->Location = System::Drawing::Point(110,
23);
            this->FPStextBox->Name = L"FPStextBox";
            this->FPStextBox->Size = System::Drawing::Size(57, 20);
            this->FPStextBox->TabIndex = 2;
            //
            // label2
            //
```

```
            this->label2->AutoSize = true;
            this->label2->Location = System::Drawing::Point(76, 23);
            this->label2->Name = L"label2";
            this->label2->Size = System::Drawing::Size(27, 13);
            this->label2->TabIndex = 5;
            this->label2->Text = L"FPS";
            //
            // PsnrForm
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(247, 108);
            this->Controls->Add(this->FPStextBox);
            this->Controls->Add(this->label2);
            this->Controls->Add(this->CancelButton);
            this->Controls->Add(this->button1);
            this->Name = L"PsnrForm";
            this->Text = L"YUV Properties";
            this->ResumeLayout(false);
            this->PerformLayout();

        }
#pragma endregion

    private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e) {
        managed_fps = FPStextBox->Text;
        PsnrForm::Close();
    }
    private: System::Void CancelButton_Click(System::Object^ sender,
System::EventArgs^ e) {
        PsnrForm::~PsnrForm();
        PsnrForm::Close();
    }
    };
}
//------------------

//File: PsnrForm.h
#pragma once
#include <string.h>
#include <msclr\marshal.h>
#include <msclr\marshal_cppstd.h>
namespace Project3 {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for PsnrForm
    /// </summary>
    public ref class PsnrForm : public System::Windows::Forms::Form
    {
        String^ managed_fps;
    private: System::Windows::Forms::TextBox^ FPStextBox;
```

```
    private: System::Windows::Forms::Label^ label2;

    public: String^ getFps() { return managed_fps; }

    public:
        PsnrForm(void)
        {
            InitializeComponent();
        }

    protected:

        ~PsnrForm()
        {
            if (components)
            {
                delete components;
            }
        }

    protected:
    protected:

    private: System::Windows::Forms::Button^ button1;
    private: System::Windows::Forms::Button^ CancelButton;

    protected:

    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void)
        {
            this->button1 = (gcnew System::Windows::Forms::Button());
            this->CancelButton = (gcnew
System::Windows::Forms::Button());
            this->FPStextBox = (gcnew
System::Windows::Forms::TextBox());
            this->label2 = (gcnew System::Windows::Forms::Label());
            this->SuspendLayout();
            //
            // button1
            //
            this->button1->Location = System::Drawing::Point(110,
60);
            this->button1->Name = L"button1";
            this->button1->Size = System::Drawing::Size(71, 23);
            this->button1->TabIndex = 9;
            this->button1->Text = L"OK";
            this->button1->UseVisualStyleBackColor = true;
            this->button1->Click += gcnew System::EventHandler(this,
&PsnrForm::button1_Click);
            //
```

```
            // CancelButton
            //
            this->CancelButton->Location = System::Drawing::Point(28,
60);
            this->CancelButton->Name = L"CancelButton";
            this->CancelButton->Size = System::Drawing::Size(75, 23);
            this->CancelButton->TabIndex = 10;
            this->CancelButton->Text = L"Cancel";
            this->CancelButton->UseVisualStyleBackColor = true;
            this->CancelButton->Click += gcnew
System::EventHandler(this, &PsnrForm::CancelButton_Click);
            //
            // FPStextBox
            //
            this->FPStextBox->Location = System::Drawing::Point(110,
23);
            this->FPStextBox->Name = L"FPStextBox";
            this->FPStextBox->Size = System::Drawing::Size(57, 20);
            this->FPStextBox->TabIndex = 2;
            //
            // label2
            //
            this->label2->AutoSize = true;
            this->label2->Location = System::Drawing::Point(76, 23);
            this->label2->Name = L"label2";
            this->label2->Size = System::Drawing::Size(27, 13);
            this->label2->TabIndex = 5;
            this->label2->Text = L"FPS";
            //
            // PsnrForm
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(6,
13);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(247, 108);
            this->Controls->Add(this->FPStextBox);
            this->Controls->Add(this->label2);
            this->Controls->Add(this->CancelButton);
            this->Controls->Add(this->button1);
            this->Name = L"PsnrForm";
            this->Text = L"YUV Properties";
            this->ResumeLayout(false);
            this->PerformLayout();

        }
#pragma endregion

    private: System::Void button1_Click(System::Object^ sender,
System::EventArgs^ e) {
        managed_fps = FPStextBox->Text;
        PsnrForm::Close();
    }
    private: System::Void CancelButton_Click(System::Object^ sender,
System::EventArgs^ e) {
        PsnrForm::~PsnrForm();
        PsnrForm::Close();
    }
    };
}
```

```
//------------------

//File: VideoYUV.cpp

#include "VideoYUV.h"

VideoYUV::VideoYUV(const char *f, int h, int w, int nbf, int
chroma_format)
{
    file = open(f, O_RDONLY | O_BINARY);
    if (!file) {
        fprintf(stderr, "readOneFrame: cannot open input file
(%s)\n", f);
        exit(EXIT_FAILURE);
    }
    height = h;
    width = w;
    nbframes = nbf;

    comp_height[0] = h;
    comp_width[0] = w;
    if (chroma_format == CHROMA_SUBSAMP_400) {
        comp_height[2] = comp_height[1] = 0;
        comp_width[2] = comp_width[1] = 0;
    }
    else if (chroma_format == CHROMA_SUBSAMP_420) {
        // Check size
        if (h % 2 == 1 || w % 2 == 1) {
            fprintf(stderr, "YUV420: 'height' and 'width' have to be
even numbers.\n");
            exit(EXIT_FAILURE);
        }

        comp_height[2] = comp_height[1] = h >> 1;
        comp_width[2] = comp_width[1] = w >> 1;
    }
    else if (chroma_format == CHROMA_SUBSAMP_422) {
        // Check size
        if (w % 2 == 1) {
            fprintf(stderr, "YUV422: 'width' has to be an even
number.\n");
            exit(EXIT_FAILURE);
        }

        comp_height[2] = comp_height[1] = h;
        comp_width[2] = comp_width[1] = w >> 1;
    }
    else {
        comp_height[2] = comp_height[1] = h;
        comp_width[2] = comp_width[1] = w;
    }
    comp_size[0] = comp_height[0] * comp_width[0];
    comp_size[1] = comp_height[1] * comp_width[1];
    comp_size[2] = comp_height[2] * comp_width[2];

    size = comp_size[0] + comp_size[1] + comp_size[2];

    data = new imgpel[size];
    luma = data;
    chroma[0] = data + comp_size[0];
    chroma[1] = data + comp_size[0] + comp_size[1];
```

```
}

VideoYUV::~VideoYUV()
{
    delete[] data;
    close(file);
}

bool VideoYUV::readOneFrame()
{
    imgpel *ptr_data = data;

    for (int j = 0; j<3; j++) {
        int read_size = comp_width[j];
        if (read_size <= 0)
            continue;
        for (int i = 0; i<comp_height[j]; i++) {
            if (read(file, ptr_data, static_cast<size_t>(read_size))
!= read_size) {
                fprintf(stderr, "readOneFrame: cannot read %d bytes
from input file, unexpected EOF.\n", read_size);
                return false;
            }
            ptr_data += read_size;
        }
    }
    return true;
}

void VideoYUV::getLuma(cv::Mat& local_luma, int type)
{
    cv::Mat tmp(height, width, CV_8UC1, this->luma);
    if (type == CV_8UC1) {
        tmp.copyTo(local_luma);
    }
    else {
        tmp.convertTo(local_luma, type);
    }
}
//------------------

//File: VideoYUV.h

#pragma once
#ifndef VideoYUV_h
#define VideoYUV_h

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <opencv2/core/core.hpp>

#ifdef _WIN32
#include <io.h>
#endif /* _WIN32 */


#ifndef _WIN32
#define O_BINARY 0
#endif /* _WIN32 */
```

```
typedef unsigned char imgpel;

// Chroma subsampling format definitions
enum ChromaSubsampling {
    CHROMA_SUBSAMP_400 = 0,
    CHROMA_SUBSAMP_420 = 420,
    CHROMA_SUBSAMP_422 = 2,
    CHROMA_SUBSAMP_444 = 3
};

class VideoYUV {
public:
    VideoYUV(const char *file, int height, int width, int nbframes,
int chroma_format);
    ~VideoYUV();
    // Read one frame
    bool readOneFrame();
    // Get the luma component
    // readOneFrame() needs to be called before getLuma()
    void getLuma(cv::Mat& luma, int type = CV_8UC1);
private:
    int file;       // file stream
    int nbframes;       // number of frames
    int height;     // height
    int width;      // width
    int comp_height[3]; // height in specific component
    int comp_width[3];  // width in specific component

    int size;       // number of samples
    int comp_size[3];   // number of samples in specific component

    imgpel *data;       // data array
    imgpel *luma;       // pointer to luma
    imgpel *chroma[2];  // pointers to chroma
};

#endif
//------------------
```

Bibliography

[1] D. Grigorios, IMPLEMENTATION OF HEVC (H.265) VIDEO ANALYSIS TOOL, Lamia, 2017.

[2] "HM 16.15 reference software," [Online]. Available: https://hevc.hhi.fraunhofer.de/.

[3] "Microsoft Visual Studio 19," [Online]. Available: https://visualstudio.microsoft.com/.

[4] "OpenCV Open Source Computer Vision Library," [Online]. Available: https://opencv.org/.

[5] P. J. Deitel and H. J. Deitel, C How to Program Fifth Edition, New Jersey: Pearson Education, 2007.

[6] B. Stroustrup, The C++ Programming Language Fourth Edition, Addison-Wesley, 2013.

[7] "draw.io," [Online]. Available: app.diagrams.net.

[8] D. S. R. Sruthi S., "Video Compression – from Fundamentals to H.264 and H.265 Standards," *International Journal Of Engineering And Computer Science ISSN:2319-7242*, vol. 4, no. 7, pp. 13468-13473, July 2015.

[9] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. -. K. Tan and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards-Including High Efficiency Video Coding (HEVC)," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 22, no. 12, pp. 1669-1684, 2012.

[10] I. E. G. Richardson, H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia, Wiley, 2003.

[11] J.-R. O.-J. H. W. Gary J. Sullivan, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 22, no. 12, pp. 1649-1668, DECEMBER 2012.

[12] M. B. J. S. Vivienne Sze, High Efciency Video Coding (HEVC) - Algorithms and Architectures, Springer, 2014.

[13] T. Wiegand, G. J. Sullivan, G. Bjøntegaard and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 13, no. 7, pp. 560-576, 2003.

[14] "x265 HEVC encoder," [Online]. Available: http://x265.org/.