Dizajn i implementacija modula za obradu videa uživo i pohranu informacija o detektiranim artefaktima

Berečić, Filip

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:494697

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-07-13**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

DIZAJN I IMPLEMENTACIJA MODULA ZA OBRADU VIDEA UŽIVO I POHRANU INFORMACIJA O DETEKTIRANIM ARTEFAKTIMA

Diplomski rad

Filip Berečić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 21.09.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Filip Berečić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 776 R, 12.10.2015.
OIB studenta:	----------
Mentor:	Doc.dr.sc. Mario Vranješ
Sumentor:	Jelena Vlaović
Sumentor iz tvrtke:	Kristijan Vučković
Predsjednik Povjerenstva:	Doc.dr.sc. Ratko Grbić
Član Povjerenstva:	Jelena Vlaović
Naslov diplomskog rada:	Dizajn i implementacija modula za obradu videa uživo i pohranu informacija o detektiranim artefaktima
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno dizajnirati i implementirati modul za obradu video materijala uživo i spremanje video zapisa s artefaktima i informacija o artefaktima. Analizu i spremanje video zapisa s detektiranim artefaktima potrebno je izvršavati u stvarnom vremenu. Također je potrebno realizirati programsko sučelje za komunikaciju aplikacije za analizu videa s implementiranim modulom. Modul treba zapisivati informacije o artefaktima koji se pojavljuju tijekom analize videa. Na temelju informacija o pojavljenim artefaktima potrebno je snimiti video zapise koji prikazuju dio video materijala u kojemu su se pojavili artefakti, kako bi se isti mogli pregledavati u nekoj drugoj aplikaciji. (sumentor Kristijan Vučković, Institut RT-RK Osijek, Cara Hadrijana 10b) (sumentor Jelena Vlaović)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	21.09.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2017.

Ime i prezime studenta:	Filip Berečić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 776 R, 12.10.2015.
Ephorus podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Dizajn i implementacija modula za obradu videa uživo i pohranu informacija o detektiranim artefaktima**

izrađen pod vodstvom mentora Doc.dr.sc. Mario Vranješ

i sumentora Jelena Vlaović

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD 1

2. VIDEO MATERIJALI 2

2.1. Modeli boja 2

2.2. Sažimanje videa 4

2.3. Artefakti 7

2.4. Izvori video materijala 11

2.5. FFmpeg 13

3. MODUL ZA OBRADU ARTEFAKATA PRONAĐENIH U IZOBLIČENOM VIDEO SIGNALU 15

3.1. Dizajn modula 15

3.2. API za komunikaciju s modulom 18

3.3. Zapisna datoteka 21

4. POHRANJIVANJE VIDEO ZAPISA S DETEKTIRANIM ARTEFAKTIMA 23

4.1. Izbor video okvira za pohranu 23

4.2. Kodiranje i spremanje video zapisa 23

4.3. Rezultati korištenja modula 24

5. ZAKLJUČAK 28

LITERATURA 29

SAŽETAK 31

ŽIVOTOPIS 32

1. UVOD

Kontrola kvalitete videa važna je stavka u televizijskoj industriji. Jedan od načina kontrole kvalitete video zapisa je analiza video materijala raznim algoritmima za pronalazak artefakata koji se mogu pojaviti tijekom sažimanja ili prijenosa video materijala. Takva analiza osigurava da će pružatelji TV usluga svojim korisnicima isporučiti kvalitetnu uslugu. Neki od artefakata koji se mogu pojaviti u video materijalima su artefakt stvaranja blokova u slici, artefakt gubitka paketa, artefakt smrzavanje slike i dr.

U tu svrhu razvijeni su posebni algoritmi koji imaju mogućnost pronalaska takvih artefakata na danim video materijalima. Ti algoritmi se u obliku priključaka dodaju u posebno razvijene aplikacije koje se koriste za analizu video materijala. Analiza se najčešće odvija u stvarnom vremenu, na video materijalima uživo, koji se preko posebnih uređaja za dohvaćanje slike dohvaćaju s nekog izvora kao što je npr. *set-top box* (STB).

Kako bi se iz artefakata pronađenih tijekom analize mogli izvući neki zaključci o kvaliteti usluge, važno je dobiti detaljnije informacije o artefaktima, kao što su njihov tip, vrijeme pojavljivanja, trajanje, lokacija. Još je bolje ako se mogu priskrbiti i dijelovi video zapisa koji sadrže te artefakte.

U drugom poglavlju dana je teorijska podloga o video zapisima općenito, kao i ona o najčešćim artefaktima u videu. Treće poglavlje opisuje modul za obradu pronađenih artefakata – njegov dizajn, API (engl. *application programming interface*) i ispis rezultata obrade. U četvrtom poglavlju opisan je proces izbora video okvira koje je potrebno spremiti, proces kodiranja te samog spremanja video zapisa. Uz to, prikazani su i cjelokupni rezultati analize te primjer aplikacije za vizualizaciju pronađenih artefakata.

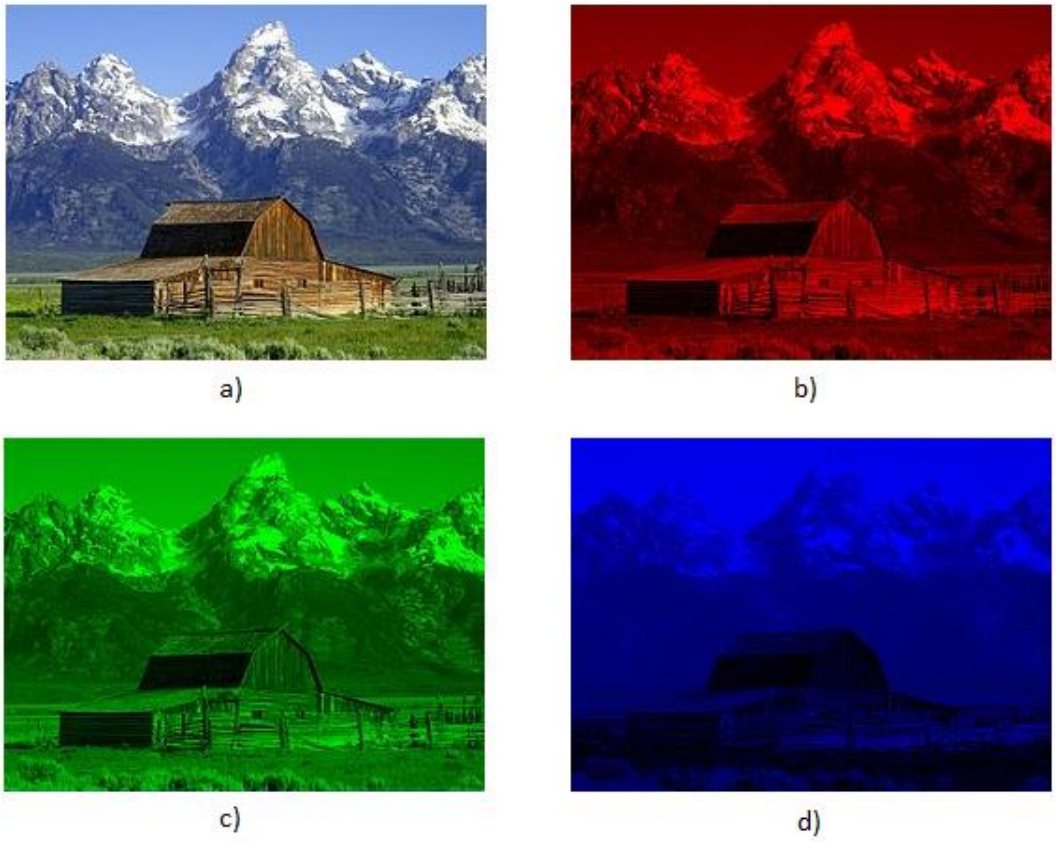
2. VIDEO MATERIJALI

Video zapis se sastoji od niza slika koje se izmjenjuju velikom brzinom, što ljudskom oku daje dojam da se slika pomiče. Da bi se stvorila ta iluzija, potrebna je izmjena najmanje 24 slike u sekundi (engl. *frames per second - FPS*). Svaka slika, još zvana i video okvir, sastoji se od elemenata slike. Broj elemenata slike jednog video okvira dobije se množenjem širine i visine okvira izraženih u broju elemenata slike. Svaki element slike zastupljen je određenom količinom bita, što ovisi o modelu boja koji se koristi.

2.1. Modeli boja

2.1.1. RGB

RGB (*engl. R - red, G - green, B - blue*) je aditivni model boja u kojem R predstavlja zastupljenost crvene, G zelene i B zastupljenost plave boje (slika 2.1.). RGB model koristi se za prikazivanje slika u elektroničkim sustavima kao što su televizori i računala [1]. Obično se koristi osam bita za predstavljanje jedne komponente boje, odnosno ukupno 24 bita po elementu slike za sve tri komponente boje.



Sl. 2.1. Slika u boji (a) prikazana svim trima komponentama boje (b) crvena komponenta boje (c) zelena komponenta boje (d) plava komponenta boje [2].

2.1.2. YUV

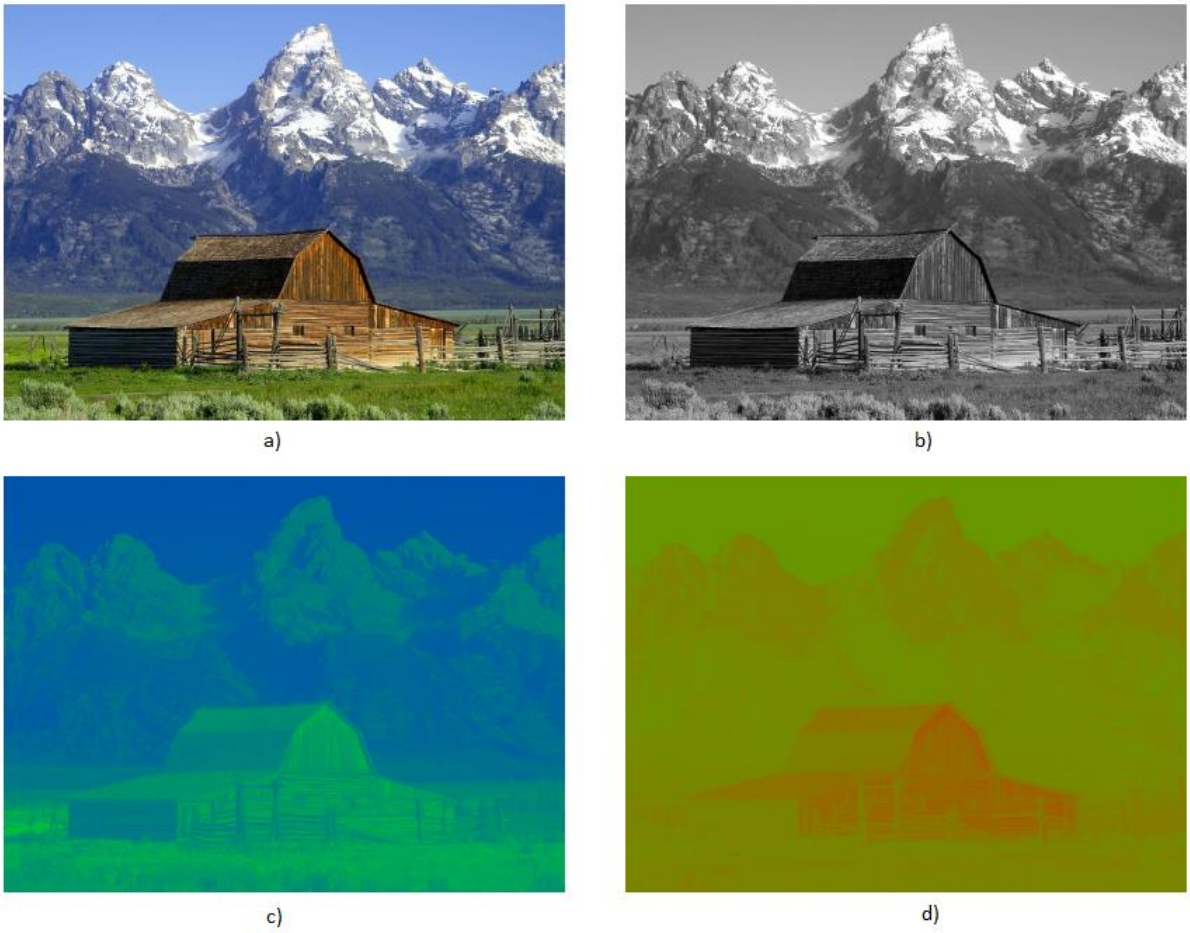
YUV (engl. *Y – luma; U, V – chroma*) je model boja koji uzima u obzir činjenicu da je ljudsko oko osjetljivije na svjetlinu nego na boje [3]. YUV model se sastoji od tri komponente: Y – luminantna komponenta te U i V – krominantne komponente. Y komponenta nosi informaciju o svjetlini, a U i V komponente informaciju o bojama (slika 2.2.).

RGB model se u YUV može pretvoriti sljedećim formulama [4]:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad (2\text{-}1)$$

$$U = -0.147 * R - 0.289 * G + 0.436 * B \quad (2\text{-}2)$$

$$V = 0.615 * R - 0.515 * G + 0.100 * B \quad (2\text{-}3)$$



Sl. 2.2. Slika u boji (a) prikazana svim trima komponentama (b) luminantna (Y) komponenta boje (c) krominantna *U* komponenta boje (d) krominantna *V* komponenta boje [5].

2.2. Sažimanje videa

Većina novo proizvedenih televizora danas podržava FullHD (1920 x 1080 elemenata slike) ili čak još veće rezolucije: UHD (Ultra HD) 4K (3840 x 2160 elemenata slike) ili 8K (7680 x 4320 elemenata slike). Kada bi htjeli pohraniti FullHD video od 60 minuta koji izmjenjuje 30 okvira po sekundi te je svaki element slike zapisan s 24 bita, njegovu bi veličinu mogli izračunati na sljedeći način:

$$veličina\ videa = 1920 * 1080 * 24\,bita * 30\,fps * 60\,s * 60\,min = \sim 670\,GB$$

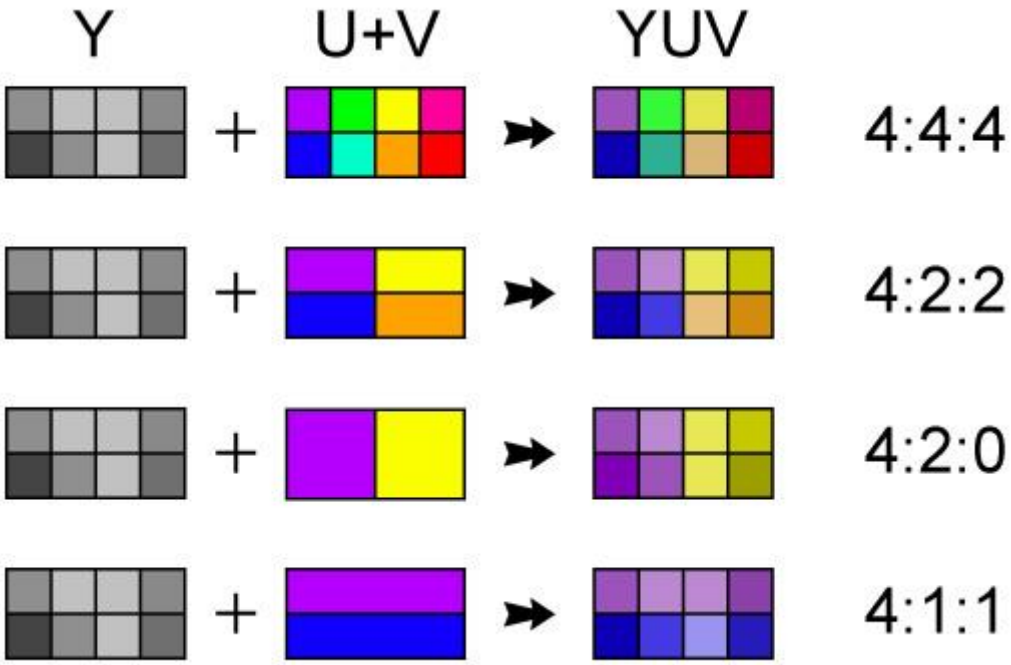
Vidimo kako bi za pohranu tog video zapisa bilo potrebno oko 670 gigabajta (GB), što je jako velika količina podataka, pa čak i za moderne diskove od nekoliko terabajta.

Kako bi se veličina video datoteka smanjila te kako bi one postale pogodnije za pohranu i prijenos, u potpunosti se iskorištava YUV model boja te ljudska percepcija svjetlosti i boja uvođenjem poduzorkovanja krominantnih komponenata [6]. Postoji nekoliko tipova poduzorkovanja krominantnih komponenata boje:

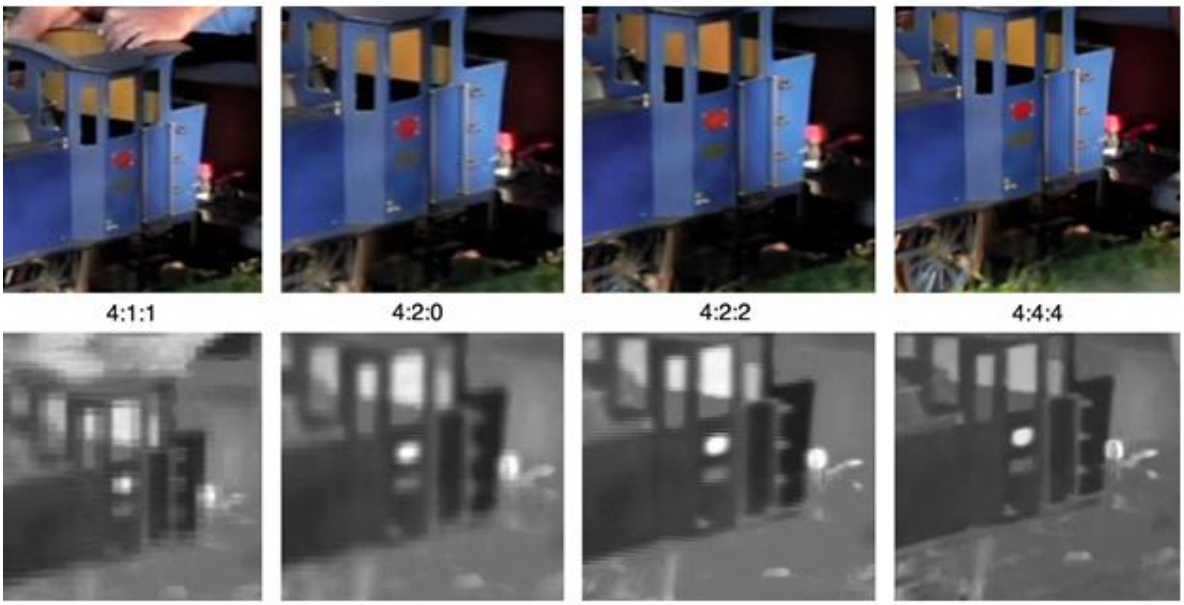
- 4:4:4 – nema poduzorkovanja; sve tri YUV komponente imaju istu stopu uzorkovanja.
- 4:2:2 – rezolucija krominantnih komponenata horizontalno se dvostruko smanjuje, čime se štedi 33.3% prostora u odnosu na okvir u kojem boja nije poduzorkovana.
- 4:2:0 – rezolucija krominantnih komponenata horizontalno i vertikalno se dvostruko smanjuje; ušteda prostora je 50 %.
- 4:1:1 – rezolucija krominantnih komponenata horizontalno se četverostruko smanjuje; štednja prostora u odnosu na nekomprimirani okvir je 50 %.

Primjer ovih vrsta poduzorkovanja može se vidjeti na slici 2.3.

Slika 2.4. prikazuje razliku između 4:1:1, 4:2:0, 4:2:2 i 4:4:4 formata poduzorkovanja. Gornji red slika prikazuje kompletnu sliku, a donji red prikazuje rezoluciju krominantnih dijelova. Moguće je primijetiti da je razlika u kvaliteti između slika gotovo pa neprimjetna – tek je na granicama između različitih boja kod slika s većim poduzorkovanjem krominantnih komponenata moguće primijetiti manje degradacije.



Sl. 2.3. Vrste poduzorkovanja krominantnih komponenata boje kod YUV video zapisa [7].



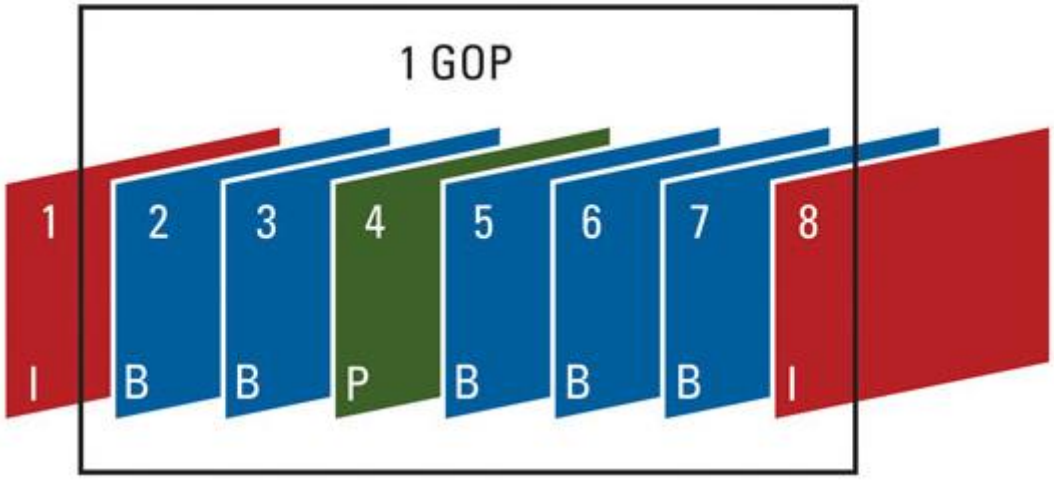
Sl. 2.4. Primjer rezultata poduzorkovanja [8].

Osim poduzorkovanja u YUV modelu boja, dodatno sažimanje videa dobiva se primjenom postupka kompresije, koji nekoliko uzastopnih okvira pretvaraju u skup okvira (engl. *group of pictures – GOP*) [9].

Tipovi okvira u GOP-u su:

- I okvir (engl. *intracoded frame*) – okvir koji se kodira neovisno o drugim okvirima. Svaki GOP počinje I okvirom. Kako je on neovisno kodiran, mogućnosti za sažimanje kompresijom kod I okvira su manje nego kod ostalih tipova okvira.
- P okvir (engl. *predictivecoded frame*) – P okvir pohranjuje samo promjene u odnosu na prethodno kodirane okvire (I ili P) te je stoga veća mogućnost njegova sažimanja nego kod I okvira.
- B okvir (engl. *bipredictive coded frame*) – B okvir pohranjuje promjene u odnosu na prethodne i sljedeće okvire (I ili P). Stupanj sažimanja najveći je kod B okvira, ali i složenost njihova računanja također.

Primjer skupa okvira može se vidjeti na slici 2.5.



Sl. 2.5. Skup okvira (GOP) [10].

2.2.1. Vrste kompresije

Postoje dvije različite vrste kompresije: kompresija s gubicima (engl. *lossy*) i kompresija bez gubitaka (engl. *lossless*) [11]. Obje vrste kompresije imaju svoje prednosti i nedostatke te se koriste sukladno traženim zahtjevima.

Kada se podaci kodiraju postupcima s gubicima, dio podataka se trajno gubi, što znači da podaci koji se kodiraju neće biti isti onima koji će se dobiti dekodiranjem [12]. Ovakav način kodiranja omogućuje drastično smanjenje podataka koje je potrebno prenijeti od izvora do odredišta pa se najčešće primjenjuje za distribuciju audio i video materijala. Suvremeni kodeci su toliko napredni te mogu postići drastično smanjenje veličine audio i video materijala uz neprimjetne gubitke sadržaja.

Kodiranje bez gubitaka [13] omogućuje potpunu rekonstrukciju komprimiranih podataka. Cijena toga je dosta veća veličina komprimirane datoteke te se zato komprimiranje bez gubitaka koristi samo onda kada je potrebna potpuna rekonstrukcija podataka. Najpoznatiji primjeri datoteka komprimiranih bez gubitaka su ZIP, PNG, GIF i dr.

Kodiranje bez gubitaka u ovome radu kasnije se koristi kako bi se spriječilo nastajanje dodatnih artefakata uslijed sažimanja video okvira na kojima se već nalaze artefakti.

2.2.2. *FF video codec 1*

FFV1 [14] („*FF video codec 1*“) je kodek razvijen u sklopu FFmpeg projekta. Koristi se za kodiranje bez gubitaka. Testiranja su pokazala [15] da FFV1 kodek postiže najbolje rezultate u uštedi podataka, ali i u brzini kodiranja. Zbog toga, FFV1 je odabran kao kodek koji se u ovom radu koristi za kodiranje video okvira na kojima su pronađeni artefakti. Detaljnije o ovome bit će rečeno u pod-poglavlju 4.2.

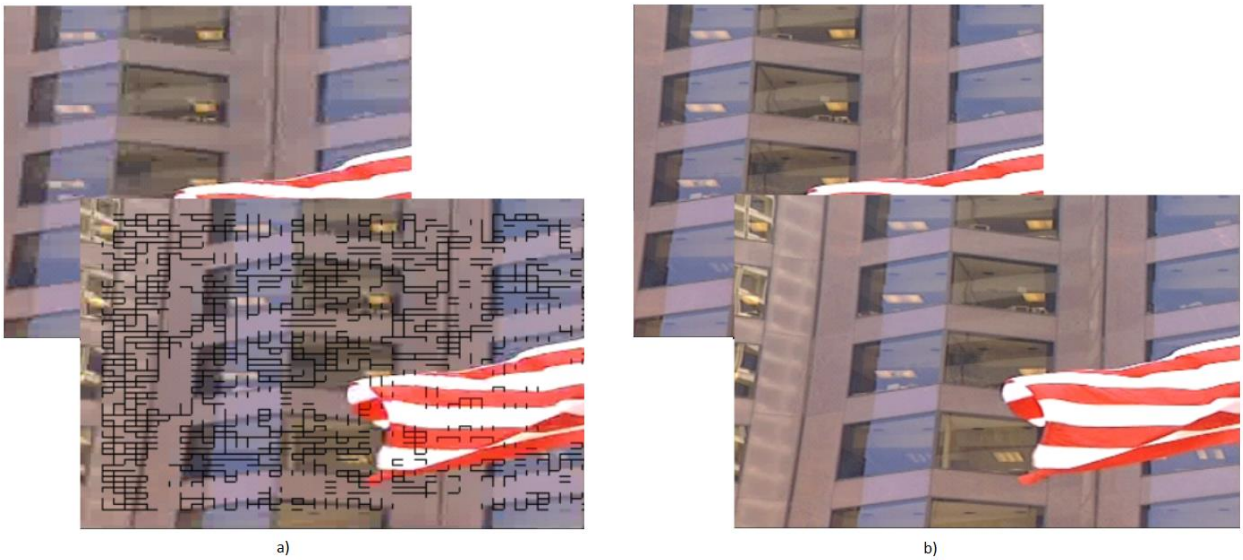
2.3. Artefakti

Korištenjem previsoke razine sažimanja ili pak loše dizajniranog kodeka, može doći do pojave neželjenih smetnji na video okvirima. Takve smetnje nazivamo video artefaktima. Oni mogu biti mali i neprimjetni, ali ako se pojavljuju u većim količinama, ili ako utječu na cijeli video okvir, mogu jako narušiti iskustvo gledanja video sadržaja.

Osim prilikom kodiranja odnosno dekodiranja, artefakti se mogu pojaviti i prilikom prijenosa video materijala. Neki od najčešćih artefakata koji se pojavljuju su stvaranje blokova, smrzavanje slike, artefakt gubitka paketa, kidanje (engl. *tearing*) slike i dr.

2.3.1. Stvaranje blokova

Artefakt stvaranja blokova (engl. *blocking*) najčešći je artefakt jer se pojavljuje kada se koristi kodiranje s gubitkom i kodeci koji rastavljaju video okvire na blokove koji se potom zasebno obrađuju. On nastaje kada je razina sažimanja previsoka pa se na dekodiranom okviru može primijetiti diskontinuitet u svjetlini ili boji na granicama blokova [16]. Na slici 2.6. može se vidjeti primjer video okvira s pristunim artefaktom stvaranja blokova.



Sl. 2.6. Primjer okvira pod utjecajem artefakta stvaranja blokova (a) okvir s artefaktom i isti okvir s označenim blokovima (b) okvir bez artefakta stvaranja blokova uz pripadni okvir s (ne)označenim artefaktima [16].

2.3.2. Smrzavanje slike

Artefakt smrzavanja slike (engl. *freezing*) nastaje prilikom prijenosa video materijala uživo uslijed iznenadnog prestanka primanja novih video okvira. To dovodi do toga da se nastavlja prikazivati zadnji pristigli video okvir sve dok ne pristigne novi. Artefakt smrzavanja slike najčešće je posljedica gubitka paketa, ali može se dogoditi da nastane i tijekom sažimanja [16].

2.3.3. Artefakt gubitka paketa

Artefakt gubitka paketa (engl. *packet loss*) pojavljuje se kada dođe do greške prilikom prijenosa video materijala. Pošto je za slanje jednog video okvira potrebno više paketa, može se dogoditi da se neki od njih izgube u prijenosu, što dovodi do pojave artefakata pravokutnog oblika (u I okvirima) s krivim sadržajem na mjestima izobličenja [16]. Primjer artefakta gubitka paketa može se vidjeti na slici 2.7.

2.3.4. Gubitak slike

Gubitak slike (engl. *image absence*) je artefakt sličan smrzavanju slike. Razlika je u tome što se kod smrzavanja slike za vrijeme nedolazaka novih okvira prikazuje zadnji pristigli okvir, a kod gubitka slike obično se prikazuje jednobojni (najčešće crni) okvir [16]. Primjer se može vidjeti na slici 2.8.

Sl. 2.7. Primjer video okvira s artefaktom gubitkom paketa [16].

Sl. 2.8. Primjer artefakta gubitka slike [16].

2.3.5. Šum

Šum (engl. *noise*) na video okviru nastaje uslijed nasumične promjene vrijednosti elemenata slike. Može se uočiti na individualnim elementima slike koji su drugačije boje od onih u okolini [16]. Primjer šuma može se vidjeti na slici 2.9.

Sl. 2.9. Usporedba (a) izvornog video okvira (b) video okvira pod utjecajem šuma [16].

2.3.6. Kidanje slike

Do kidanja slike (engl. *tearing*) dolazi kada je brzina osvježavanja video materijala različita od brzine osvježavanja uređaja za prikaz videa. Prepoznaje se po „pokidanim“ linijama na video okviru [16]. Primjer artefakta kidanja slike može se vidjeti na slici 2.10.

Sl. 2.10. Usporedba (a) izvornog okvira (b) okvira s artefaktom kidanja slike [16].

2.3.7. Nestajanje boje i svjetlosti

Nestajanje svjetlosti (engl. *brightness fading*) primjećuje se prilikom mjerenja na luminantnoj komponenti, a nestajanje boje (engl. *color fading*) prilikom mjerenja na krominantnim komponentama. Ukoliko izmjerene vrijednosti odstupaju od onih očekivanih, detektira se anomalija u video okviru [16]. Primjer se može vidjeti na slici 2.11.

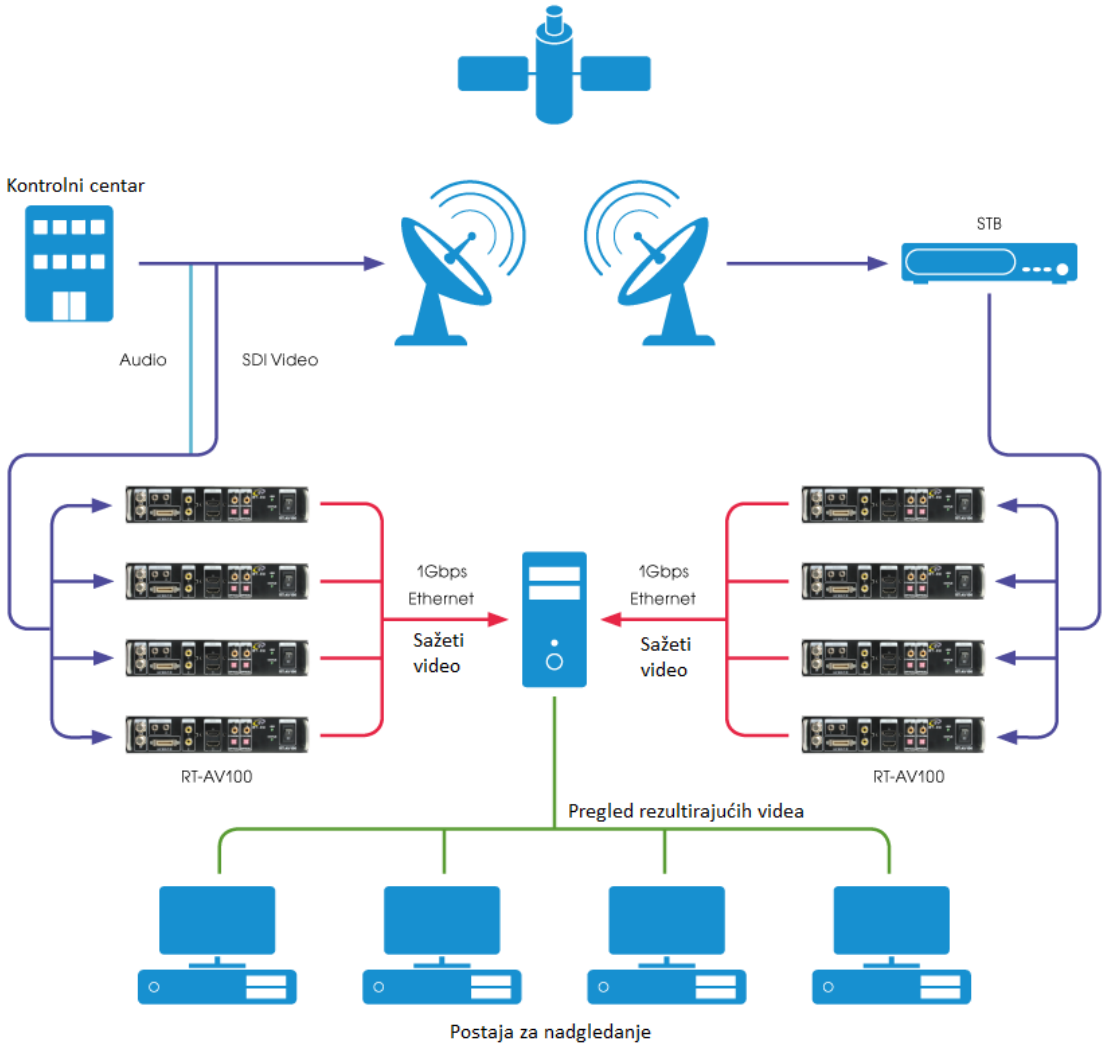
Sl. 2.11. Primjer odstupanja svjetline (a) izvorni okvir (b) okvir pod utjecajem artefakta odstupanja svjetline [16].

Za pronalazak artefakata u video okvirima koriste se specifični algoritmi koji su napravljeni za traženje artefakata okvir po okvir, tj. na bazi jednog okvira, jer se analiza obično provodi na video okvirima koji pristižu u stvarnom vremenu sa specijalnog uređaja za dohvaćanje slike s nekog izvora.

2.4. Izvori video materijala

Izvor video materijala koji se analiziraju može biti video datoteka pohranjena na disku, video uživo preko interneta, video signal prije odašiljanja krajnjim korisnicima ili na strani prijemnika televizijskog signala. Na slici 2.12. može se vidjeti jedan sustav za analizu kvalitete video signala prije i poslije prijenosa.

Uređaj pomoću kojega se dohvaća video naziva se *grabber*. On može dohvaćati video s različitih analognih ili digitalnih ulaza te podržava različite standarde prijenosa i formata video materijala. Najčešće se koristi za dohvaćanje video materijala prilikom testiranja STB-ova, DVD ili Blu-Ray uređaja. Na slici 2.13. može se vidjeti primjer uređaja za dohvaćanje video materijala.



Sl. 2.12. Primjer scenarija analize prije i poslije odašiljanja video signala [17].

Sl. 2.13. Primjer uređaja za dohvaćanje video materijala [18].

2.5. FFmpeg

FFmpeg [19] je skup alata za rad s multimedijskim podacima koji se sastoji od biblioteka i programa zasnovanih na dotičnim bibliotekama. FFmpeg se može koristiti za razne operacije s multimedijskim podacima, kao što su kodiranje, dekodiranje, transkodiranje, multipleksiranje, demultipleksiranje, filtriranje i reproduciranje gotovo svih vrsta multimedijskih sadržaja. Podržava različite multimedijske formate, od onih najstarijih pa sve do najmodernijih, pa i one koji su razvijeni u privatnim kompanijama i za koje ne postoji javno dostupna specifikacija. FFmpeg je napisan na način da bude prenosiv te je stoga dostupan za sve najkorištenije operacijske sustave (Linux, Max OS X, Microsoft Windows, BSD, Solaris i dr.) i za različite procesorske arhitekture [20].

Programi od kojih se FFmpeg sastoji nabrojani su u nastavku:

- *ffmpeg* – koristi se za transkodiranje multimedijskih datoteka.
- *ffserver* – poslužitelj koji se koristi za emitiranje multimedije uživo.
- *ffplay* – jednostavni program koji se koristi za reproduciranje multimedijskih datoteka.
- *ffprobe* – jednostavni program za analizu multimedijskih datoteka.

FFmpeg-ovi programi se isporučuju u obliku izvršnih datoteka za ciljani operacijski sustav i arhitekturu, a pokreću se i koriste iz naredbenog retka (engl. *command line*). Svi parametri i postavke za korištenje programa opisani su u službenoj dokumentaciji [21].

Programi za rad s multimedijskim datotekama napisani su pomoću FFmpeg-ovih biblioteka koje su napisane u C programskom jeziku koristeći C99 standard. Biblioteke se isporučuju kao statičke u kombinaciji s datotekama zaglavlja, ili kao dinamičke, ovisno o tome koriste li se za razvoj ili za pokretanje programa koji ih koriste.

U nastavku su navedene FFmpeg-ove biblioteke:

- *libavcodec* – sadrži funkcije i strukture za kodiranje i dekodiranje zvuka i videa. Podržava veliki broj kodeka i integralni je dio samog FFmpeg-a, ali i brojnih drugih programa, kao što su mpv, VLC, Google Chrome, OpenCV i dr.
- *libavformat* – pruža funkcije za multipleksiranje i demultipleksiranje tokova videa, zvuka i podnaslova i rad s raznim multimedijskim formatima. Osim toga, podržava i rad s nekoliko različitih protokola za ulaz odnosno izlaz multimedije, kao što su UDP, TCP, HTTP, FTP, SCTP, BluRay i dr.

- *libavfilter* – služi za filtriranje zvuka odnosno videa. Sadrži nekoliko različitih filtera, izvora i izlaza.
- *libswscale* – koristi se za reskaliranje video zapisa i pretvaranje između različitih formata elemenata slike.
- *libswresample* – služi za operacije sa zvukom, kao što su promjena formata, promjena stope frekvencije uzorkovanja i slično.
- *libavdevice* – pruža funkcionalnost dohvaćanja i prikazivanja multimedije sa, odnosno na, razne multimedijske ulazno/izlazne uređaje.
- *libavutil* – sadrži funkcije koje pomažu prilikom pisanja programa za više platformi, funkcije za generiranje slučajnih brojeva, kriptografiju, dodatne matematičke funkcije i slično.

U ovom radu korištene su tri FFmpeg-ove biblioteke: *libavcodec* za kodiranje video okvira, *libavformat* za spremanje kodiranih video okvira u video datoteku te *libavutil* za pomoćne funkcije.

3. MODUL ZA OBRADU ARTEFAKATA PRONAĐENIH U IZOBLIČENOM VIDEO SIGNALU

Modul za obradu pronađenih artefakata u izobličenom video signalu napisan je u programskom jeziku C++ i isporučuje se u obliku statičke biblioteke (*.lib* ekstenzija na Microsoft Windows operacijskim sustavima te *.a* ekstenzija na operacijskim sustavima koji su potekli od Unix-a) i datoteke zaglavlja (*.h* ekstenzija, engl. *header file*) koja predstavlja API modula. Datoteka zaglavlja uključuje se u izvorni kod programa na mjestu gdje se koristi API modula.

Statička se biblioteka, za razliku od dinamičke, u izvršnu datoteku aplikacije dodaje prilikom povezivanja prevedenog izvornog koda aplikacije za analizu video materijala. Uz male promjene, moguće je statičku biblioteku pretvoriti u dinamičku koja se učitava prilikom pokretanja ili korištenja programa, što bi smanjilo veličinu izvršne datoteke aplikacije, ali i omogućilo selektivno korištenje modula, ovisno o njegovoj dostupnosti u sustavu.

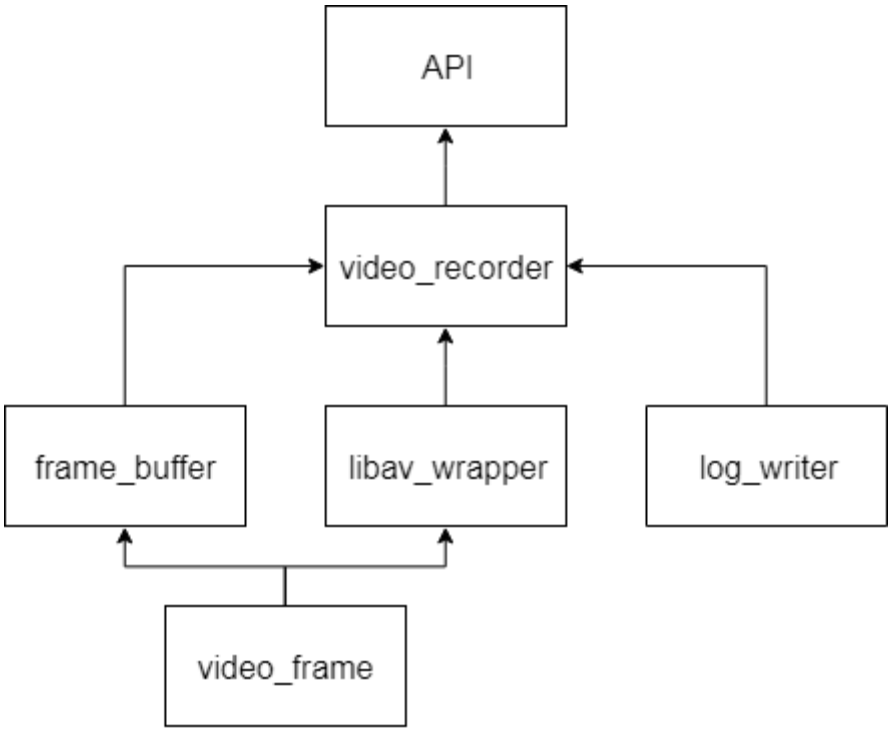
Kako analiza video materijala može trajati nekoliko dana, svako zapisivanje informacija o artefaktima i spremanje video zapisa na disk mora se izvršavati paralelno s analizom, jer u suprotnom ne bi bilo dovoljno radne memorije za pohranjivanje svih video okvira kada bi se video zapisi spremali po završetku analize.

3.1. Dizajn modula

Modul za obradu pronađenih artefakata sastoji se od pet pod-modula:

- *video*_frame - modul koji definira video okvir.
- *frame_buffer* - modul međuspremnika za video okvire.
- *libav_wrapper* - modul apstrakcije FFmpeg biblioteka.
- *log_writer* - modul za pisanje zapisne datoteke.
- *video_recorder* - glavni modul za snimanje koji koristi sve ostale module.
- *API/main* - modul u kojem su implementirane funkcije API-ja.

Na slici 3.1. može se vidjeti ovisnost između pod-modula.



Sl. 3.1. Hijerarhija korištenja pod-modula u stvorenom modulu za obradu artefakata pronađenih u izobličenom video signalu

3.1.1. Pod-modul *video_frame*

Pod-modul *video_frame* sadrži definiciju video okvira. U njemu se nalazi istoimena struktura koja opisuje video okvir. Struktura *video_frame* je osnova za pod-module *frame_buffer* i *libav_wrapper* jer oni moraju znati definiciju video okvira kako bi ga na ispravan način mogli pohraniti i dobiti informacije o podacima pohranjenim u njemu. Informacije koje se pohranjuju za svaki video okvir su:

- *ID* – identifikacijski broj svakog video okvira. Potreban je kako bi se mogao identificirati video okvir u kojemu se pojavljuje artefakt.
- *width* – širina video okvira. Potrebno je znati dimenzije svakog video okvira jer one mogu biti različite.
- *height* – visina video okvira.
- *data* – pokazivač na polje u kojemu se nalaze vrijednosti svakog elementa slike Y komponente izvornog videa. Veličina polja određena je dimenzijama video okvira.

3.1.2. Pod-modul *frame_buffer*

Pod-modul *frame_buffer* igra ulogu međuspremnika (engl. *buffer*) za video okvire. Kada pristigne novi video okvir, on se kopira u međuspremnik. Međuspremnik ima ograničeni broj video okvira koji mogu biti u njemu u isto vrijeme kako bi se osigurala što manja potrošnja radne memorije.

Nakon što međuspremnik dosegne svoj maksimalni kapacitet, svako dodavanje novog video okvira rezultirati će brisanjem prvog, tj. najstarijeg okvira u međuspremniku. Za pohranu video okvira koristi se struktura dvostruko povezani popis koja osigurava konstantno vrijeme umetanja novih okvira u međuspremnik i brisanja starih okvira s početka. Također, osigurava i to da će svi video okviri u međuspremniku, između prvog i zadnjeg, uvijek imati istu lokaciju u memoriji, što omogućava istovremeno pisanje i čitanje u, odnosno iz, međuspremnika.

Umetanje novog video okvira u međuspremnik vrši se metodom *push* klase *frame_buffer* koja kao parametre prima ID video okvira te njegovu širinu i visinu. Kao rezultat vraća *true*, ako je video okvir uspješno umetnut u međuspremnik, te *false*, ako nije. Prvo se vrši provjera je li pokazivač na podatke izvornog video okvira postavljen. Zatim, vrši se provjera je li međuspremnik na maksimalnom kapacitetu – ako je, najstariji video okvir se briše. Zatim se vrši umetanje novog okvira uz kontrolu pogreške. Ako se prilikom umetanja dogodi iznimka, to znači da video okvir nije uspješno umetnut u međuspremnik jer nema dovoljno slobodne memorije. Ako je okvir uspješno umetnut, podaci iz izvornog okvira video materijala koji se analizira kopiraju se u novo umetnuti okvir. Na slici 3.2. dan je kod metode za umetanje novog video okvira u međuspremnik.

3.1.3. Pod-modul *libav_wrapper*

Pod-modul *libav_wrapper* je *wrapper* oko funkcionalnosti *FFmpeg*-ovih biblioteka *libavcodec, libavformat i libavutil.* Ovaj pod-modul pruža samo jednu funkciju – *record_video* koja prima pokazivač na prvi video okvir i na zadnji video okvir te vrši snimanje svih video okvira koji se nalaze u zatvorenom intervalu između prvog i zadnjeg, u jedan video zapis. *Wrapper* oko *libavcodec, libavformat i libavutil* biblioteka koristi se zbog očuvanja kompatibilnosti između različitih verzija biblioteka jer neke funkcije koje se koriste u verziji FFmpega korištenoj u ovom radu su zastarjele u novijim verzijama, tako da ukoliko se ukaže potreba za prijelaz na noviju verziju, neće biti potrebno promijeniti cijeli *video_recorder* već samo *libav_wrapper*.

3.1.4. Pod-modul *log_writer*

Pod-modul *log_writer* je koji je zadužen za pisanje informacija o analizi videa u zapisnu datoteku. Informacije o pronađenim artefaktima zapisuju se nakon spremanja video zapisa s tim artefaktima kako bi korisnik zapisne datoteke znao u kojim video zapisima se nalaze dani artefakti. Iako se sve informacije o artefaktima zapisuju prilikom njihove pojave, odnosno nakon spremanja video zapisa, zapisna datoteka neće biti ispravno formatirana sve dok se ne završi analiza i pravilno se zatvori JSON dokument.

```
12  bool frame_buffer::push(frame_id_t frame_id, frame_size_t width, frame_size_t height)
13  {
14      if (data_source_ptr_ == nullptr) {
15          return false;
16      }
17
18      if (frames_.size() == max_buffer_size_) {
19          // if buffer is at full capacity, remove first frame
20          frames_.pop_front();
21      }
22
23      try {
24          frames_.emplace_back(frame_id, width, height);
25      }
26      catch (const std::bad_alloc&) {
27          // out of memory -> frame constructor threw -> new frame wasnt inserted
28          return false;
29      }
30
31      std::copy(data_source_ptr_, data_source_ptr_ + width * height, frames_.back().data);
32      return true;
33  }
```

Sl. 3.2. Kod metode za umetanje novog video okvira u međuspremnik

3.1.5. Pod-modul *video_recorder*

Pod-modul *video*_recorder je glavni pod-modul u kojemu se nalazi sva logika vezana za spremanje video zapisa i informacija o pronađenim artefaktima. On koristi ostale pod-module i njihove unutarnje API-je. Pod-modul *video_recorder* dobiva informacije o svakom okviru i svim pronađenim artefaktima te ih dalje prosljeđuje ostalim modulima zaduženim za njih.

3.1.6. API/main

API/main pod-modul predstavlja implementaciju samog modula. U njemu su implementirane funkcije API-ja. Osim implementacija funkcija, sadrži i pomoćne varijable za pohranu algoritama i njihovih pragova te provjeru ispravnosti. U njemu su definirane i unaprijed zadane vrijednosti modula koje je moguće promijeniti putem posebne funkcije iz API-ja.

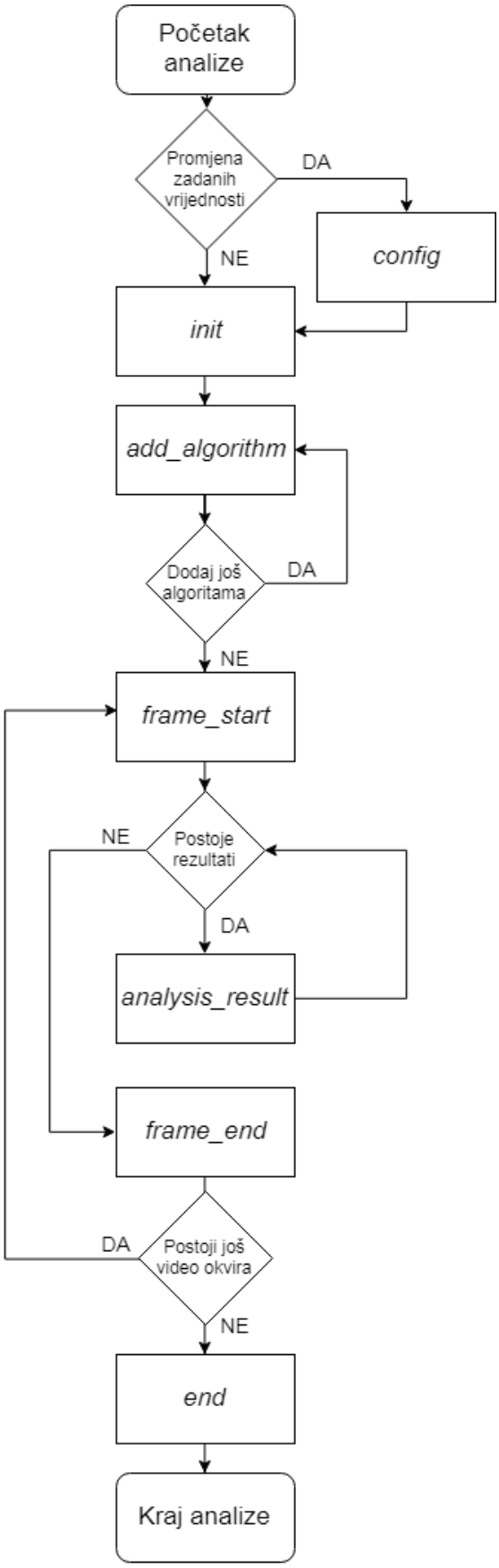
3.2. API za komunikaciju s modulom

API za komunikaciju s modulom sastoji se od sedam funkcija koje su deklarirane u datoteci zaglavlja, a implementirane u *API/main* pod-modulu. Osim uključivanja zaglavlja, aplikacija u svoj izvorni kod posebnom direktivom prevoditelja uključuje i statičku biblioteku kako bi znala gdje se u memoriji nalaze funkcije deklarirane u zaglavlju.

U nastavku su opisane funkcije koje definiraju API:

- *config* – funkcija za promjenu unaprijed zadanih vrijednosti modula kao što su broj video okvira u sekundi videa – fps, broj video okvira koje treba spremiti prije i poslije okvira s pronađenim artefaktom te veličina međuspremnika za video okvire. Funkciju *config* potrebno je pozvati prije inicijalizacije modula funkcijom *init*, ukoliko se odluči da unaprijed zadane parametre treba promijeniti.
- *init* – funkcija koja služi za inicijalizaciju modula. Njoj je potrebno proslijediti informacije o tome u koji direktorij se spremaju video zapisi i informacije o artefaktima, memorijsku lokaciju na kojoj počinje video okvir te maksimalnu veličinu pojedinog video okvira.
- *add_algorithm* – funkcija koja se koristi za dodavanje informacija o korištenim algoritmima za pronalazak artefakata tijekom analize. Za svaki algoritam potrebno je, osim imena, dati i prag za osjetljivost algoritma tj. da li je pronađeni artefakt dovoljno značajan da bi ga se trebalo zabilježiti i spremiti. Modul pohranjuje naziv i prag svakog algoritma kako bi kasnije, na temelju rezultata analize, mogao odlučiti o tome koje video okvire treba spremiti.
- *frame_start* – funkcija koja signalizira da je pristigao novi video okvir te je spreman za kopiranje u međuspremnik. Ova funkcija kao parametar mora primiti veličinu video okvira jer ona može biti drugačija za svaki okvir. Nakon poziva ove funkcije, modul pokreće proces kopiranja video okvira u međuspremnik.
- *analysis_result* – funkcija koja šalje rezultate analize trenutnog video okvira. Za svaki rezultat, šalje naziv algoritma kojim je video okvir analiziran te brojčani rezultat analize. Zatim modul vrši provjeru prelazi li rezultat analize video okvira tim algoritmom prag - ako ga prijeđe, video okvir se dodaje na popis video okvira koje je potrebno spremiti u video zapis.
- *frame_end* – funkcija koja signalizira kraj analize trenutnog video okvira.
- *end* – funkcija koja signalizira kraj cjelokupne analize. Nakon pozivanja ove funkcije modul završava pisanje zapisne datoteke i video zapisa te se vraća u prvotno stanje kako bi se mogla započeti nova analiza.

Dijagram toka rada s modulom može se vidjeti na slici 3.3.



Sl. 3.3. Dijagram toka korištenja API-ja modula za obradu artefakata pronađenih u izobličenom video signalu

3.3. Zapisna datoteka

Informacije o svim pronađenim artefaktima zapisuju se u zapisnu datoteku kako bi se kasnije mogle pročitati i detaljnije analizirati. Zapisna datoteka zapisana je u JSON (engl. *JavaScript Object Notation*) formatu što omogućuje jednostavno čitanje, kako ljudima tako i programskim jezicima.

Informacije koje se zapisuju u zapisnu datoteku dane su u nastavku:

- *start_time* – vrijeme početka analize.
- *fps* – broj video okvira u sekundi.
- *artifact_types* – vrste artefakata koji se mogu pojaviti tijekom analize.
- *artifacts* – popis svih artefakata koji su se pojavili tijekom analize. Za svaki artefakt koji se pojavio, zapisano je slijedeće:
    - *type* – vrsta artefakta.
    - *frame_id* – broj (ID) video okvira na kojemu se pojavio artefakt.
    - *video_file* – ime video zapisa u koji je spremljen dio izvornog videa s artefaktom odnosno artefaktima.
- *number_of_frames* – broj analiziranih video okvira.
- *end_time* – vrijeme završetka analize.

Vrijeme početka i kraja analize zapisani su po ISO8601 standardu jer većina programskih jezika podržava automatsko čitanje istoga. Vrste artefakata koje se mogu pojaviti u video materijalima zapisane su da bi aplikacija koja čita i vizualizira zapisnu datoteku unaprijed znala za koliko se vrsta artefakata treba pripremiti vizualizacija.

Primjer zapisa zapisne datoteke može se vidjeti na slici 3.4.

Iz primjera se može vidjeti kako je analiza trajala 60 sekundi te je analizirano 960 video okvira, a na sedam okvira pronađeni su artefakti. Iako je pronađeno sedam artefakata, postoji samo četiri video zapisa u koje su spremljeni artefakti. Razlog tomu je taj što se artefakte, koji su u blizini jedan drugome za određeni broj video okvira (predefiniran u modulu, ali ga je moguće i promijeniti *config* funkcijom API-ja), sprema u isti video zapis.

```
{
   "start_time": "2017-07-11T11:24:41",
   "fps": "16",
   "artifact_types": ["blocking", "freezing", "packet loss"],
   "artifacts": [
      {"type": "blocking",    "frame_id": "8",   "video_file": "1.avi"},
      {"type": "freezing",    "frame_id": "10",  "video_file": "1.avi"},
      {"type": "packet loss", "frame_id": "50",  "video_file": "2.avi"},
      {"type": "freezing",    "frame_id": "88",  "video_file": "3.avi"},
      {"type": "packet loss", "frame_id": "889", "video_file": "4.avi"},
      {"type": "blocking",    "frame_id": "890", "video_file": "4.avi"},
      {"type": "packet loss", "frame_id": "891", "video_file": "4.avi"}
   ],
   "number_of_frames": "960",
   "end_time": "2017-07-11T11:25:41"
}
```

Sl. 3.4. Primjer zapisa zapisne datoteke

Zapisna datoteka namijenjena je za čitanje od strane aplikacije za vizualizaciju artefakata. Na temelju pročitanih informacija aplikacija za vizualizaciju stvara vremenski prikaz na kojemu se iscrtavaju svi artefakti koji su se pojavili tijekom analize. Klikom na pojedini artefakt moguće je dobiti više informacija o tom artefaktu, kao što su točno vrijeme pojave i identifikacijski broj video okvira originalnog video materijala na kojemu se pojavljuje. Također, klikom na pojedini artefakt započinje se reprodukcija video zapisa u kojemu je spremljen artefakt. Međutim, ovaj dio nije bio predmet ovog diplomskog rada, već je moguć kao njegova nadogradnja.

4. POHRANJIVANJE VIDEO ZAPISA S DETEKTIRANIM ARTEFAKTIMA

4.1. Izbor video okvira za pohranu

Za svaki algoritam korišten u pronalasku artefakata na video materijalima postoji prag koji označava je li rezultat analize pojedinog video okvira algoritmom dovoljno značajan (prelazi prethodno zadani prag) da bi se analizirani video okvir spremio u video zapis. Prag svakog algoritma postavlja se prilikom njegovog dodavanja funkcijom *add_algorithm*.

Nakon umetanja novo pristiglog video okvira u međuspremnik, vrši se provjera rezultata algoritma te se odlučuje o tome hoće li on biti pohranjen u video zapis. Ako je video okvir potrebno spremiti, dodaje se na popis okvira koje je potrebno spremiti u zapisnu datoteku i video zapis. Osim samog video okvira na kojem su pronađeni jedan ili više artefakata, potrebno je spremiti i određeni broj video okvira prije i poslije njega. Taj je broj unaprijed definiran, ali moguće ga je promijeniti funkcijom *config*. Spremanje okolnih video okvira radi se zbog pružanja mogućnosti promatranja okolnih okvira onoga okvira na kojemu se pojavio artefakt, kako bi se eventualno mogli donijeti neki zaključci o nastajanju artefakta.

U slučaju da se oko video okvira pod utjecajem artefakata pojavi još jedan okvir pod utjecajem artefakata, te da je udaljenost između ta dva okvira manja ili jednaka broju okvira koje je potrebno spremiti prije odnosno poslije okvira, tada se oba okvira spremaju u jednu datoteku.

4.2. Kodiranje i spremanje video zapisa

Spremanje video zapisa u kojima su prikazani artefakti pronađeni analizom originalnih video materijala radi se zbog pružanja mogućnosti naknadnog pregledavanja nastalih artefakata. Kako prilikom spremanja video zapisa s artefaktima ne bi došlo do pojave novih artefakata koji bi mogli biti smetnja prilikom pregledavanja pronađenih artefakata, potrebno je koristiti kodiranje bez gubitka podataka. Na taj način, dekodirani video zapis biti će isti kao onaj originalni. Kodek koji se u ovom radu koristi za kodiranje video okvira je FFV1, koji je detaljnije opisan u pod-poglavlju 2.2. FFmpeg-ova biblioteka *libavcodec* korištena je za proces kodiranja pojedinih video okvira. Bibliotekom *libavformat* kodirani video okviri pišu se u video datoteku.

Na početku korištenja FFmpeg-ovig biblioteka potrebno je pozvati funkciju *av_register_all* kojom se biblioteke inicijaliziraju. Zatim je potrebno otvoriti kontekst izlaznog formata funkcijom *avformat_alloc_output_context2*. Funkcijom *avcodec_find_encoder* pronalazi se željeni kodek.

Nakon pronalaska kodeka, datoteka u koju će se video spremiti otvara se funkcijom *avio_open2*. Funkcijom *avformat_new_stream* dodaje se novi tok u datoteku. Prije nego se video okvir kodira, potrebno ga je pripremiti za kodiranje. Kako video okviri koji na analizu dolaze s *grabber*-a mogu biti različitih dimenzija, one okvire koji su manjih dimenzija od referentnih potrebno je prilagoditi kako bi se mogli kodirati zajedno s referentnim. Prilagođavanje se vrši tako da se okviru manjih dimenzija povećaju širina i visina, a prazan prostor oko rubova popuni se crnom bojom.

Kodiranje okvira vrši se funkcijom *avcodec_encode_video2*. Kodirani video okviri zapisuju se u video pakete koji se potom funkcijom *av_write_frame* pišu u video datoteku. Kako bi izvođači video zapisa znali koji kodek treba koristiti za dekodiranje video paketa, prije njihovog pisanja u datoteku treba napisati zaglavlje u kojemu se nalaze informacije o korištenom kodeku, formatu i slično. Pisanje zaglavlja radi se funkcijom *av_write_header*. Osim na početku, isto je potrebno napraviti i na kraju, funkcijom *av_write_trailer*. Na kraju, potrebno je sve resurse osloboditi funkcijama *av_frame_free*, *avcodec_close* i *avio_close*.

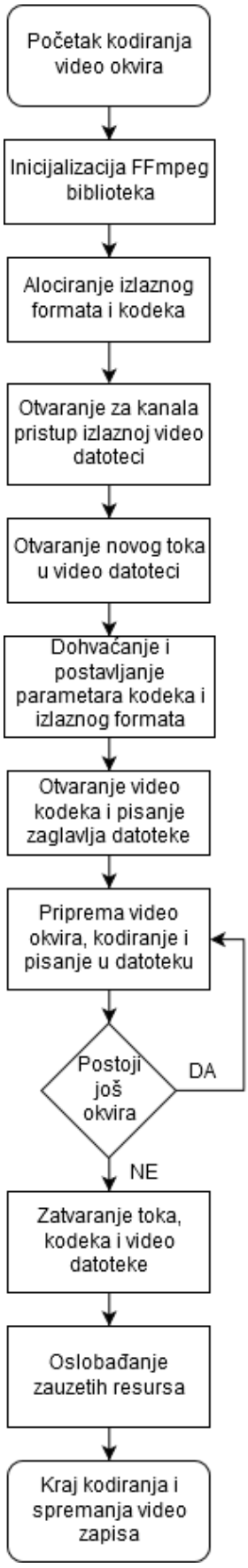
Na slici 4.1. može se vidjeti dijagram toka procesa kodiranja video okvira i spremanja video datoteke korištenjem FFmpeg-ovih biblioteka.

Pošto je kodiranje video okvira procesorski zahtjevan posao, ono može na duži period blokirati nit na kojoj se poziva API modula, što bi dovelo do nemogućnosti izvršavanja analize video materijala u stvarnom vremenu. Zbog toga je kodiranje video okvira i spremanje video zapisa potrebno izvršavati u zasebnoj niti.

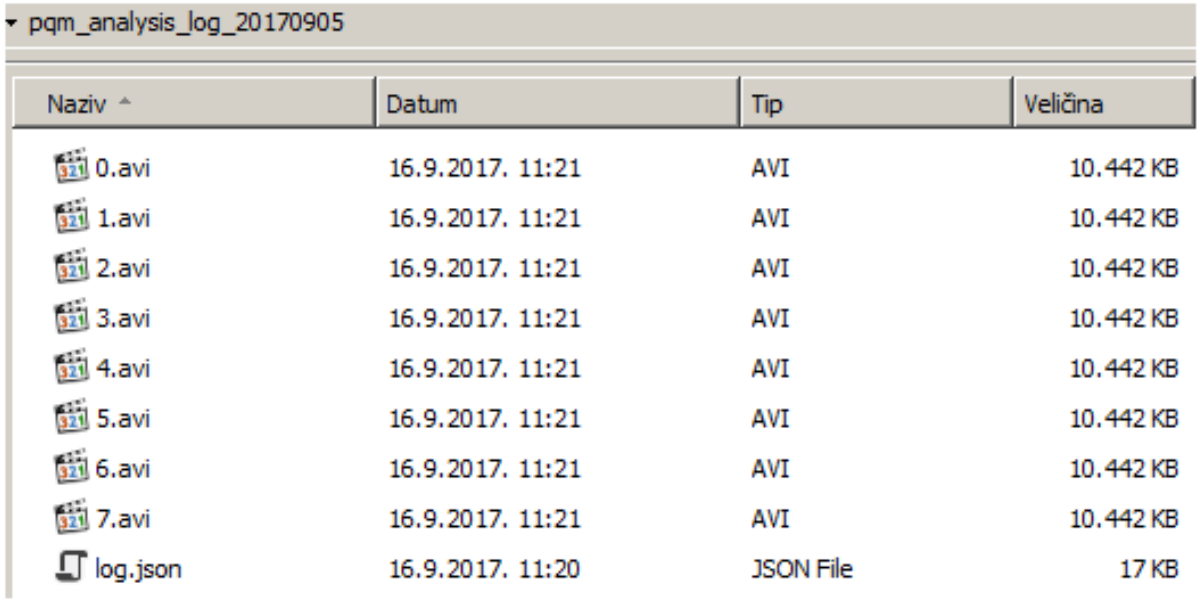
4.3. Rezultati korištenja modula

Rezultati korištenja modula tijekom analize video materijala su zapisna datoteka opisana u poglavlju 3.3. te određen broj video zapisa u kojima se nalaze video okviri s artefaktima. Na slici 4.2. može se vidjeti kako izgleda direktorij u kojemu su pohranjeni zapisna datoteka i video zapisi s artefaktima.

Zapisnu datoteku je moguće otvoriti u nekom od programa za pregledavanje teksta te na taj način pogledati popis artefakata koji su se pojavili tijekom analize. Druga opcija je zapisnu datoteku otvoriti u nekom programskom jeziku koji podržava čitanje JSON formata pa iz nje pročitati sve informacije, ispisati ih na drugačiji način ili napraviti neke vizualizacije – grafove i dr.



Sl. 4.1. Proces kodiranja video okvira i spremanja video zapisa korištenjem FFmpeg-ovih biblioteka

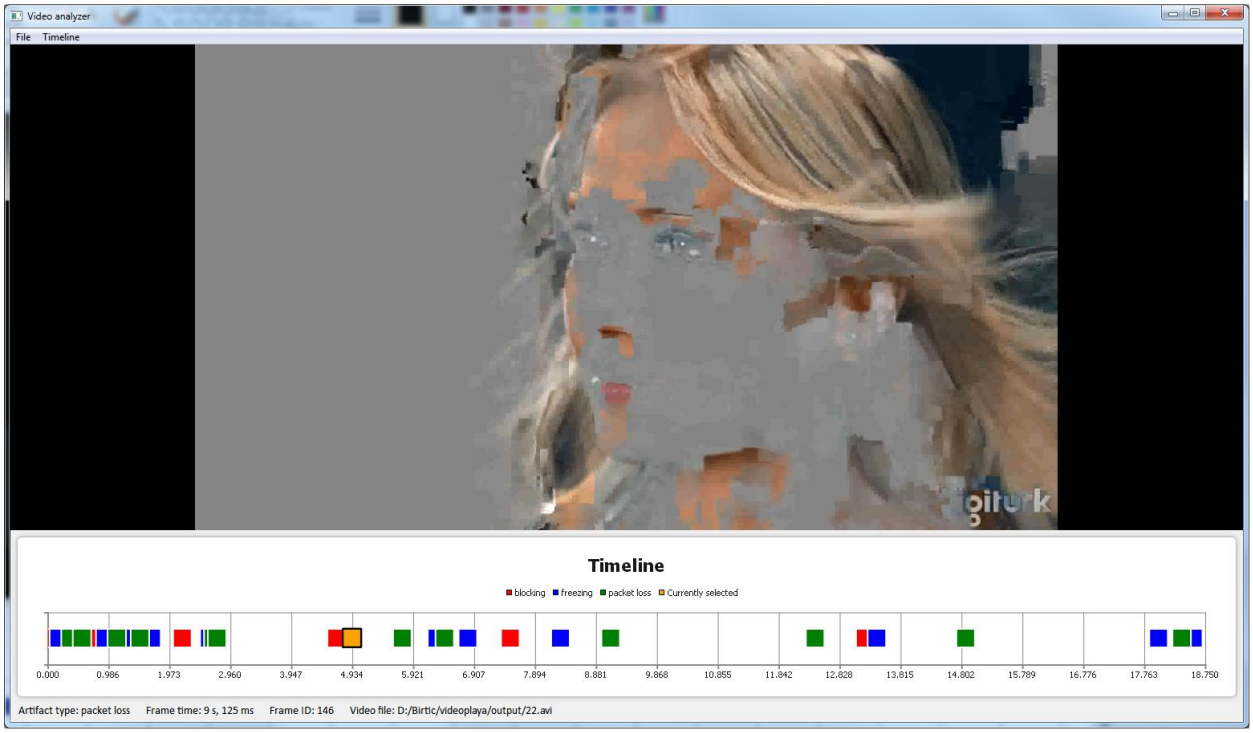


Sl. 4.2. Primjer izgleda direktorija nakon završene analize kreiranim modulom

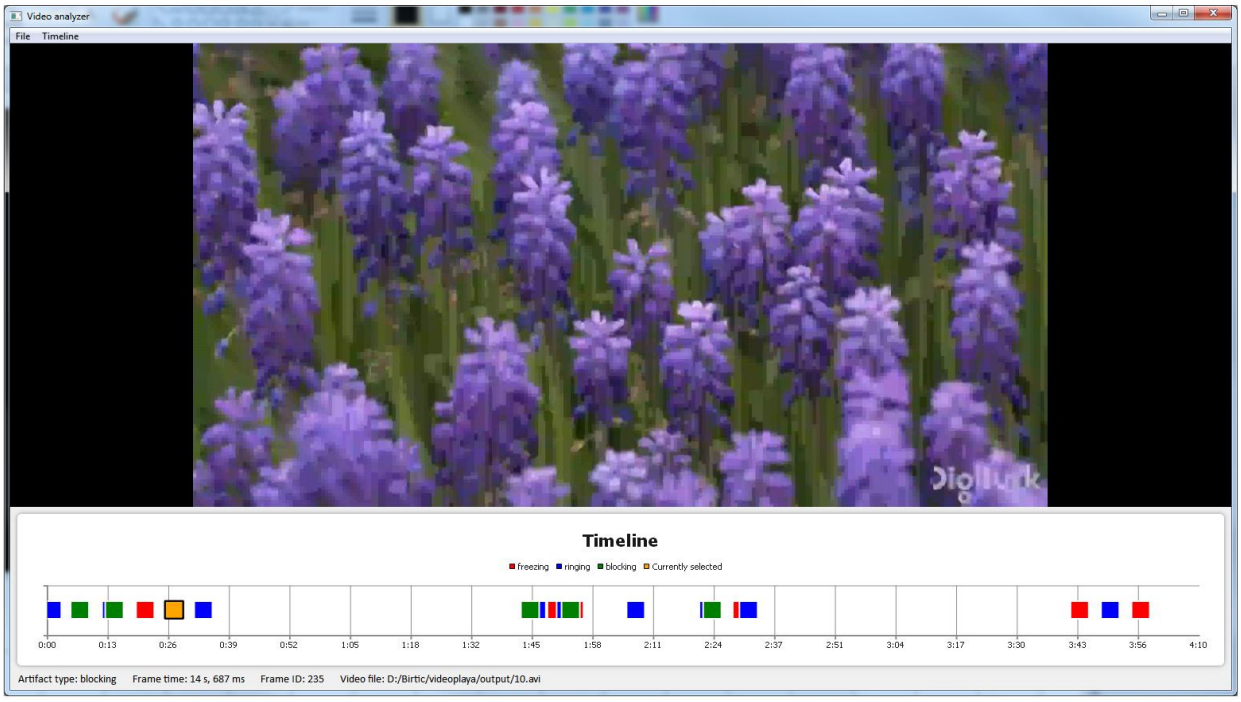
Video zapise je također moguće otvoriti i pogledati u nekom od programa za reprodukciju, ali je uvjet da oni podržavaju dekodiranje videa kodiranih FFV1 kodekom. Dekodiranje FFV1 kodiranog videa podržano je u svim poznatijim programima za reprodukciju video zapisa [22], kao što su VLC, MPlayer, mpv, Windows Media Player, Media Player Classic i dr.

Osim ručnog pregledavanja, zapisnu datoteku moguće je otvoriti i posebnom aplikacijom za vizualizaciju artefakata. Aplikacija je posebno razvijena za čitanje zapisne datoteke opisane u ovome radu, a kreirana je u sklopu drugog diplomskog rada [23]. Izgled aplikacije može se vidjeti na slikama 4.3 i 4.4.

Aplikacija se sastoji od dva glavna dijela: vremenska traka s popisom artefakata kroz vrijeme te dijela za izvođenje video zapisa. Klikom na pojedini artefakt prikažu se informacije o njemu: koji je tip artefakta, kada je nastao, njegov ID te ime video zapisa u kojem je spremljen video okvir s tim artefaktom. Također, klikom na artefakt pokreće se i reprodukcija video zapisa s tim artefaktom. Reprodukciju video zapisa moguće je pauzirati i pomicati u malim koracima, kako bi se artefakt mogao pomnije pogledati.



Sl. 4.3. Primjer aplikacije za vizualizaciju artefakata prilikom prikaza video zapisa s pronađenim artefaktom gubitka paketa



Sl. 4.4. Primjer aplikacije za vizualizaciju artefakata prilikom prikaza video zapisa s pronađenim artefaktom stvaranja blokova

5. ZAKLJUČAK

U diplomskom radu izrađen je modul za aplikaciju za analizu video materijala koji pohranjuje informacije o artefaktima pronađenim tijekom analize videa. Osim informacija o artefaktima, pohranjuju se i video zapisi koji prikazuju pronađene artefakte. Definiran je API pomoću kojega aplikacija za analizu komunicira s modulom.

Rezultate rada modula moguće je primijeniti za pomniju analizu video okvira na kojima se pojavljuju artefakti kako bi se iz njih mogle izvući korisne informacije o razlozima nastajanja artefakata – jesu li nastali zbog sažimanja, zbog smetnji prilikom prijenosa preko mreže, zraka ili nekog drugog medija ili zbog nekog drugog razloga. Informacije o nastalim artefaktima važne su kod proučavanja problema u procesu stvaranja video sadržaja i u procesu prijenosa istoga, jer je svim proizvođačima i distributerima video sadržaja u cilju pružiti što bolju uslugu svojim korisnicima.

LITERATURA

[1] „Light and Color“. [Na internetu]. Dostupno na: http://lodev.org/cgtutor/color.html#The_RGB_Color_Model_. [Pristupljeno: 18-ruj-2017].

[2] „components_02.png (PNG Image, 676 × 598 pixels)“. [Na internetu]. Dostupno na: http://www.johnloomis.org/ece564/notes/colors/components/html/components_02.png. [Pristupljeno: 15-ruj-2017].

[3] C. Wright, „YUV Colorspace“. [Na internetu]. Dostupno na: http://softpixel.com/~cwright/programming/colorspace/yuv/. [Pristupljeno: 18-ruj-2017].

[4] „YUV to RGB Conversion“. [Na internetu]. Dostupno na: http://www.fourcc.org/fccyvrgb.php. [Pristupljeno: 18-ruj-2017].

[5] „320px-Barn-yuv.png (320×957)“, 06-ruj-2017. [Na internetu]. Dostupno na: https://upload.wikimedia.org/wikipedia/commons/thumb/2/29/Barn-yuv.png/320px-Barn-yuv.png. [Pristupljeno: 06-ruj-2017].

[6] „Chroma Subsampling: 4:4:4 vs 4:2:2 vs 4:2:0“, *Rtings.com*. [Na internetu]. Dostupno na: http://www.rtings.com/tv/learn/chroma-subsampling. [Pristupljeno: 18-ruj-2017].

[7] „316-F6-Chroma-Subsampling-SECONDARY_1.png (PNG Image, 800 × 364 pixels)“. [Na internetu]. Dostupno na: https://www.videomaker.com/sites/videomaker.com/files/styles/vm_image_token_lightbox/public/articles/15788/316-F6-Chroma-Subsampling-SECONDARY_1.png?itok=ZJHCI6oE. [Pristupljeno: 15-ruj-2017].

[8] „Colorcomp.jpg (JPEG Image, 1236 × 616 pixels)“. [Na internetu]. Dostupno na: https://upload.wikimedia.org/wikipedia/commons/0/06/Colorcomp.jpg. [Pristupljeno: 15-ruj-2017].

[9] „Final Cut Pro 7 User Manual: Video Compression“. [Na internetu]. Dostupno na: https://documentation.apple.com/en/finalcutpro/usermanual/index.html#chapter=C%26section=12. [Pristupljeno: 18-ruj-2017].

[10] „Comparing JPEG 2000 and MPEG | TvTechnology“. [Na internetu]. Dostupno na: http://www.tvtechnology.com/multiformat/0112/comparing-jpeg---and-mpeg/268018. [Pristupljeno: 15-ruj-2017].

[11] „Data Compression Explained“. [Na internetu]. Dostupno na: http://mattmahoney.net/dc/dce.html. [Pristupljeno: 18-ruj-2017].

[12] „Lossy Data Compression“. [Na internetu]. Dostupno na: https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossy/index.htm. [Pristupljeno: 18-ruj-2017].

[13] „Lossless Data Compression“. [Na internetu]. Dostupno na: https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/index.htm. [Pristupljeno: 18-ruj-2017].

[14] *The FFV1 lossless video codec specification*. FFmpeg, 2017.

[15] „Comparing video codecs and containers for archives“. [Na internetu]. Dostupno na: http://download.das-werkstatt.com/pb/mthk/info/video/comparison_video_codecs_containers.html#codec_tests. [Pristupljeno: 15-ruj-2017].

[16] „Video analysis | BBT“, [Na internetu]. Dostupno na: http://bbt.rt-rk.com/audiovideo-analysis/ [Pristupljeno: 06-ruj-2017].

[17] „RT-AV Station | BBT“, [Na internetu]. Dostupno na: http://bbt.rt-rk.com/hardware/rt-av_station/ [Pristupljeno: 06-ruj-2017].

[18] „RT-AV100 | BBT“, [Na internetu]. Dostupno na: http://bbt.rt-rk.com/hardware/rt-av100/ [Pristupljeno: 06-ruj-2017].

[19] „FFmpeg“, 12-ruj-2017. [Na internetu]. Dostupno na: https://ffmpeg.org/. [Pristupljeno: 12-ruj-2017].

[20] „About FFmpeg“, 12-ruj-2017. [Na internetu]. Dostupno na: https://ffmpeg.org/about.html. [Pristupljeno: 12-ruj-2017].

[21] „Documentation“. [Na internetu]. Dostupno na: http://ffmpeg.org/documentation.html. [Pristupljeno: 15-ruj-2017].

[22] „FFV1 - Wikipedia“. [Na internetu]. Dostupno na: https://en.wikipedia.org/wiki/FFV1#Applications_supporting_FFV1. [Pristupljeno: 18-ruj-2017].

[23] K. Birtić, „Rješenje za prikaz vremenske skale i snimljenog video sadržaja s greškama uočenim pri analizi video zapisa“, diplomski rad, Fakultet elektrotehnike, računarstva i informacijskih tehnologija, Osijek, 2017

SAŽETAK

U ovom diplomskom radu izrađen je modul za pohranjivanje informacija i video zapisa o artefaktima pronađenim analizom različitih video materijala. Artefakti koji se pronalaze analizom video materijala prosljeđuju se modulu za pohranjivanje pomoću definiranog API-ja. Sve funkcije API-ja su opisane i dan je dijagram toka koji pokazuje kako se API koristi. Opisani su svi dijelovi modula i njihova primjena te zapisna datoteka u koju se zapisuju informacije o svim pronađenim artefaktima. Objašnjen je proces izbora video okvira koji se spremaju u video zapis i proces kodiranja video okvira FFmpeg-ovim bibliotekama, te su prikazani rezultati pohrane.

Ključne riječi: kodiranje videa, analiza videa, video artefakti

DESIGN AND IMPLEMENTATION OF MODULE FOR LIVE VIDEO PROCESSING AND STORAGE OF INFORMATION ABOUT DETECTED ARTIFACTS

ABSTRACT

This paper describes design of a module used for storing information and video files of artifacts found during analysis of video materials. Artifacts that are found during the analysis are forwarded to the module using the defined API. All API functions are described and the provided diagram flow showcases usage of the API. Module parts and their usage is described as well as log file which stores information about artifacts found during analysis. The process of selecting artifact-affected video frames to encode with FFmpeg libraries and save to file is explained. Results and their visualisation is shown in an example artifact visualisation application.

Key words: video coding, video analysis, video artifacts

ŽIVOTOPIS

Filip Berečić rođen je 01.10.1992. u Osijeku. Nakon završetka osnovne škole u Ladimirevcima, upisuje Elektrotehničku i prometnu školu Osijek, smjer Tehničar za računalstvo. Završetkom srednjoškolskog obrazovanja, 2011. godine, upisuje preddiplomski studij Računarstva na Elektrotehničkom fakultetu u Osijeku. Trenutno je student na 2. godini diplomskog studija Računarstvo, smjer Informacijske i podatkovne znanosti, na sada Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. U studenome 2016. postaje stipendist Instituta RT-RK Osijek d.o.o.

_____

Filip Berečić