

УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Тина Мастиловић

МАСТЕР РАД

Имплементација видео плејера
сферичног видеа
коришћењем библиотека *FFmpeg* и *OpenCV*

Ментор: проф. др Владимир Филиповић



Београд,
2017.

Универзитет у Београду

Математички факултет

Мастер рад

Аутор: Тина Мастиловић 1041/2016

Наслов: Имплементација видео плејера сферичног видеа
коришћењем библиотека *FFmpeg* и *OpenCV*

Ментор: проф. др Владимир Филиповић
Математички факултет, Универзитет у Београду

Чланови комисије: др Јелена Граовац
Математички факултет, Универзитет у Београду
др Александар Картељ
Математички факултет, Универзитет у Београду

Датум одбране: _____

Садржај

| | | |
|-------|--|----|
| 1 | Видео плејер сферичног видео садржаја | 1 |
| 2 | Виртуелна стварност | 2 |
| 2.1 | Категорије <i>VR</i> садржаја | 3 |
| 2.1.1 | <i>VR</i> фотографија, панорамска фотографија | 3 |
| 2.1.2 | <i>VR</i> видео, сферични видео | 4 |
| 3 | Дигитални видео | 7 |
| 3.1 | Генерисање дигиталног видеа | 7 |
| 3.1.1 | Узимање узорака - узорковање | 8 |
| 3.1.2 | Простор боја | 8 |
| 4 | Компресија видео података | 11 |
| 4.1 | Компресија података | 11 |
| 4.2 | Стандарди за видео компресију | 11 |
| 4.2.1 | <i>JPEG</i> | 12 |
| 4.2.2 | <i>MPEG</i> | 12 |
| 4.2.3 | <i>ITU-H</i> серије | 13 |
| 5 | Библиотека <i>SDL</i> | 14 |
| 6 | Библиотека <i>OpenCV</i> | 15 |
| 7 | Библиотека <i>FFmpeg</i> | 16 |
| 8 | Фрејмови видеа и трансформације | 18 |
| 8.1 | Еквидистантна цилиндрична пројекција | 18 |
| 8.2 | <i>OpenCV</i> и функција <i>remap()</i> | 19 |
| 8.3 | Трансформације фрејмова, <i>cubemap</i> пројекција | 20 |
| 8.3.1 | Креирање странице коцке | 21 |
| 8.3.2 | Трансформације фрејмова, ротација сфере | 23 |
| 9 | Имплементација видео плејера | 28 |
| 9.1 | Иницијализација и учитавање улазног тока | 28 |
| 9.2 | Иницијализација кодека | 30 |
| 9.3 | <i>SDL</i> имплементација | 31 |
| 9.3.1 | <i>SDL</i> нит | 31 |
| 9.3.2 | <i>SDL</i> и испртавање фрејмова | 33 |
| 9.3.3 | <i>SDL</i> ослушкивање и обрада догађаја | 36 |

Глава 1

Видео плејер сферичног видео садржаја

Видео садржај прилагођен за репродуковање уз могућност избора правца и промене угла посматрања назива се сферични видео. Сферични видео се користи као све доминантнији вид забаве као и при имплементацији разних симулатора. Доводи до повећане интерактивности корисника, ставља корисника у центар радње и пружа му искуство “правог приступа догађају”.

Циљ рада је имплементација видео плејера сферичног видеа. Видео плејер као улазни параметар користи видео садржај чији су фрејмови у облику еквидистантне цилиндричне пројекције. Такви фрејмови се трансформишу у облик који представља правоугаону пројекцију и у таквом облику се приказују на екрану. За трансформације фрејмова користи се библиотека *OpenCV*, док се библиотека *FFmpeg* користи за руковање видео садржајем. Реализација корисничког интерфејса обезбеђена је коришћењем библиотеке *SDL*.

Глава 2

Виртуелна стварност

Виртуелна стварност (енг. *Virtual Reality - VR*) је појам који се односи на технологију која обезбеђује искуство присуства на одређеном простору без физичког (стварног) присуства. *VR* симулира присуство на одређеном месту у реалном времену.

Ова технологија описује тродимензионално, рачунарски генерисано окружење које може бити истраживано од стране корисника и у самој интеракцији са њим [1]. На тај начин корисник постаје део “виртуелног света” и омогућено му је да обавља низ радњи и рукује објектима у том свету [2].

Иако је *VR* стара идеја, реализација је била веома тешка за остваривање. За реализацију ове идеје неопходна је максимална сарадња хардвера и софтвера, који опет појединачно морају бити на веома високом нивоу.

Још је 1920. године Едвин Алберт Линк (*Edwin Albert Link*) изумео први симулатор летења који је служио за обуку пилота (енг. *the Link Trainer*). 1965. године, Ајван Садерланд (*Ivan Edward Sutherland*) је објавио рад “The ultimate display” где је описао како ће једног дана рачунар обезбедити “прозор” у виртуелни свет. Садерланд је 1968. године, уз помоћ свог студента Роберта Спrouла (*Robert F. Sproull*), креирао први *VR head-mounted display sistem - HMD*. *HMD* је био контролисан од стране рачунарског система. Овај систем је подсећао на преносиви телевизор и није био погодан за коришћење. Морао је бити причвршћен на метални сталак како би омогућио корисницима да га носе и да се безбедно крећу. Сматра се да овај изум представља “зачетак” *VR*.

Термин “виртуелна стварност” први пут је употребио Јарон Ланер (*Jaron Lanier*) 1987. године током интезивног истраживања у овој области. 1985. године, Ланер је заједно са Томасом Цимерманом (*Thomas Zimmerman*) основао *VPL Research* – фирму која се сматра пиониром у области истраживања *VR* и *3D* графике. *VPL Research* је продала прву *VR* опрему попут наочара и рукавица. У то време (деведесетих) *VR* је била веома популарна међутим због великог јаза између очекивања јавности и технолошких ограничења популарност је убрзо драстично опала. *VPL Research* је банкротирала 1990. године а њихове патенте је 1999. године купио *Sun Microsystems* [3].

1993. године, на сајму *CES - Consumer Electronic Show*, представљен је први модеран *VR* систем - *Sega VR* систем. Овај систем је поседовао технологију за праћење покрета главе, *LCD* екране и стерео звук. Ипак, остатак хардвера и софтвера који је био неопходан још није био ни близу потребног нивоа, па је тако први *Sega VR* систем остао на нивоу прототипа иако је продаја на тржишту већ била најављена [4].

Virtual Boy је био први *VR* систем који је уведен на тржиште (1995./1996. године). Међутим, овај систем је остао упамћен као један од највећих пехова компаније *Nintendo*. Наиме, лоша резолуција, лоше синхронизован приказ на екранима и неа-

декватни угао гледања нису утицали само на лоше искуство корисника већ су имали одређене негативне здравствене последице, првенствено у виду мучнине са симптомима сличним морској болести, тзв. *cutter* мучнина.

Иако *VR* није новина, популарност ове технологије драстично је порасла након што су највеће друштвене мреже *Youtube* и *Facebook* омогућили пренос и преглед *VR* садржаја на својим страницама. Гледање *VR* садржаја више није резервисано само за неколицину већ је доступно свима а могућност прегледа је доступна на разним уређајима (од монитора до *VR* наочара).

2.1 Категорије *VR* садржаја

Постоје 2 основне категорије *VR* садржаја: фотографија и видео.

VR Фотографија је фотографија са могућношћу интеракције крајњег корисника са виртуелним окружењем у којем се налази. На тај начин се постиже реалан доживљај стварности.

VR Видео, познат још као сферични видео или видео 360°, је видео који ставља корисника у само средиште догађања, тако да корисник сам одлучује шта ће гледати у одређеном тренутку.

2.1.1 *VR* фотографија, панорамска фотографија

Панорама (термин потиче из грчког језика) подразумева свеобухватни поглед или представу физичког простора. Простор који се приказује може бити слика, фотографија, цртеж или 3D модел. Панорамска фотографија приказује подручје веће од оног које стаје на фотографију снимљену стандардним објективом. Препоручује се за мултимедију и разне апликације које имају могућност приказа објектата у различитим величинама. Техника сликања панораме датира још из старе ере. Пронађени су мурали из Помпеје који представљају неки вид панораме. Картографски експерименти из доба просветитељства представљају претечу ове технике какву данас знамо. Патентирао ју је Роберт Баркер (*Robert Barker*) 1787. године под називом "Апаратура за представљање слика". У 19. веку панорама је представљала веома популаран модел за представљање пејзажа и слика са историјском тематиком. Европска публика била је одушевљена панорамом и утиском присуства који је она будила.



Слика 2.1: Панорама Лондона, Роберт Баркер, 1792. година

Убрзо је панорамска фотографија заменила сликање панораме као најчешће методе за представљање свеобухватног погледа. Кренуло се са монтажом и уклапањем више слика у једну ширу слику.

VR фотографија (360°фотографија) представља неки вид панорамске фотографије са могућношћу интеракције крајњег корисника са виртуелним окружењем у којем се налази. На тај начин се крајњем кориснику пружају информације на природнији начин него помоћу класичне слике и постиже се реалан доживљај простора. *VR* фотографија је погодна за приказивање разних догађаја, ентеријера и екстеријера. Све чешће се користе виртуелне туре за истраживање отвореног простора, музеја, галерија, хотелских соба, туристичких локација и разних неприметнине [5]. Виртуелне туре (виртуелне шетње) настају повезивањем више *VR* фотографија у једну целину. Неопходно је да се тачке са којих се снимају панорамске (*VR*) фотографије међусобно виде ради креирања узајамних веза. Свака од виртуелних тура може да представља мултимедијалну презентацију ако се у њу угради позадинска музика, слике, текст или видео који се приказују када корисник кликне на неки од посебно обележених објеката (енг. *Hot-Spot*).

360°фотографија пружа врхунску резолуцију и квалитет слике са могућношћу интеракције крајњег корисника са виртуелним окружењем у којему се налази чиме се постиже реалистичан доживљај простора, а крајњем кориснику пружају информације на много природнији начин од класичне Веб (енг. *web*) странице са slikama и описом.

Креирање панорамске фотографије

Дигитална фотографија с краја 20. века, поједностављује процес монтаже и креирања панорамске фотографије. Панорамска фотографија се креира тако што се више фотографија (са преклапајућим садржајем) комбинује и повезује у једну ширу слику (ент. *photo stitching*). Многе дигиталне камере данас имају уграђену опцију комбинања и повезивања слика за креирање панораме. Такође, постоје разни софтвери који обављају овај процес. Неки од тих софтвера су бесплатни и доступни на интернету.

2.1.2 *VR* видео, сферични видео

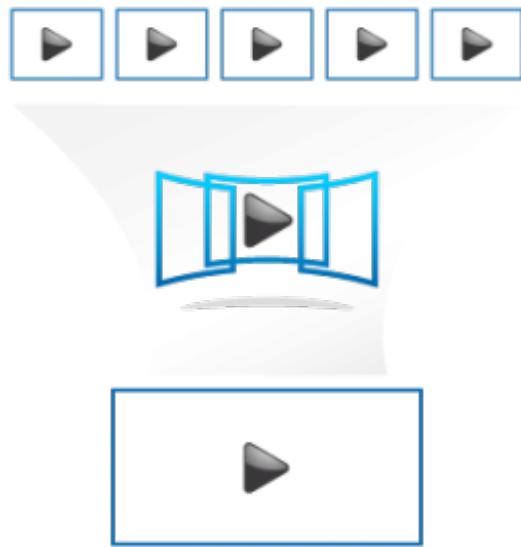
Све је више видео садржаја прилагођених пуштању уз могућност избора правца и промене угла посматрања. Овакав видео назива се сферични видео (*VR* видео, 360°видео). Појава сферичног видеа изазвала је праву револуцију у начину креирања и прегледа видео садржаја. Крајњи корисник је постављен у средиште догађања, те сам бира шта ће гледати у одређеном тренутку.

Због реалног приказа садржаја, предности *VR* видеа у односу на стандардни видео процењене су [2]:

- Дуже време прегледа видео садржаја - 28
- Већи проценат прегледа садржаја до краја - 100
- Мања цена *CPM-a* (*Cost per impression*) - 250
- Повећани *CTR* (*Click through rate*) - 800

Креирање сферичног видеа

Читаво искуство гледања видеа променило се појавом сферичног видеа. Стандарди приказивања видео садржаја веома су порасли. Сферични видео је креиран помоћу специјализованог система од неколико камера које заједно покривају и снимају све углове сцене, 360 x 180. Снимци који се направе помоћу тог система камера се затим повезују у један видео садржај (енг. *video stitching*). Овај процес подразумева комбиновање различитих видеа (чији се садржај преклапа) у циљу стварања једног видеа високе резолуције. Обавља се помоћу специјализованог рачунарског софтвера. Неки од ових софтвера су бесплатни и доступни на интернету [6][7]. Боја и контраст у сваком снимку подешава се у складу са осталим снимцима. Добијени видео покрива 360 x 180 видног поља и представља неку врсту панораме.



Слика 2.2: Комбиновање више видеа у један

Резултујући видео, добијен овим процесом покрива све углове сцене и представља неки облик пројекције сличне картографској пројекцији. Картографска пројекција је начин приказивања Земљине површине, тј. њеног елипсоида (или сфере) на раван. Пројекције које се најчешће добијају као резултат су:

- Правилна пројекција (*Rectilinear projection*)
- Цилиндрична пројекција (*Cylindrical projection*)
- Панини пројекција (*Panini projection*)
- Стереографска пројекција (*Stereographic projection*)

У овом раду подразумева се да се као улазни видео садржај добија видео који је у облику еквидистантне цилиндричне пројекције.

Еквидистантна цилиндрична пројекција (енг. *Spherical projection, Equirectangular projection*) је облик цилиндричне пројекције која приказује 360° хоризонталног и 180° вертикалног видног поља. Панораме у облику ове пројекције изгледају као да је слика умотана у сферу и да се затим та слика посматра из центра сфере. Сматра се да је ову пројекцију измислио Маринос из Тира (*Marinus of Tyre*) 100. године ПНЕ.

Глава 3

Дигитални видео

Данас је дигитални видео у великој мери део аспеката пословања, образовања и забаве. Дигитални видео је визуелна информација представљена у дискретној форми, погодна за дигитално електронско складиштење и пренос. У овом поглављу, биће описаны и дефинисани концепти дигиталног видеа: креирање дводимензионалне сцене узимањем узорка из тродимензионалне сцене и суочавање са проблемом избора система за представљање боја (RGB или $YCrCb$) [8]. Циљ видео-кодирања је да подржи видео комуникацију "прихватљивог" визуелног квалитета.

Видео слика је пројекција $3D$ сцене на дводимензионалну раван. Сама слика представља дводимензионалну пројекцију сцене у одређеном временском тренутку, док видео секвенце представљају сцену током одређеног временског периода.

Стварна визуелна сцена је континуирана и у простору и у времену. У циљу дигиталног представљања и обраде сцене неопходно је узимати узорке просторно (просторне узорке, обично на правоугаоној мрежи равни видео слике, енг. *spatial sampling*) и временске узорке (обично као низ слика или фрејмова одабраних у регуларним временским интервалима, енг. *temporal sampling*). Дигитални видео је репрезентација видео сцене добијене просторно-временским узорковањем у дигиталном облику. Сваки просторно-временски узорак (елемент слике или пиксел) се представља дигитално као један или више бројева који описују осветљеност и боју узорка.

На улазу система је, стварна визуелна сцена, добијена обичном камером и конвертована у дигитални приказ добијен узорковањем (енг. *sampled digital representation*). Овим дигиталним видео сигналом се онда може руковати у дигиталном домену на више начина, укључујући обраду, складиштење и пренос. На излазу овог система, дигитални видео сигнал се приказује на $2D$ экрану.

3.1 Генерисање дигиталног видеа

Видео је добијен помоћу камере или система камера. Најчешћи су дигитални видео системи који користе $2D$ видео добијени помоћу једне камере.

Генерисање дигиталне репрезентације видео сцене се може поделити у две фазе. Прва фаза обухвата претварање пројекције сцене у електрични сигнал. Друга фаза је фаза дигитализације која обухвата узимање узорака (просторно и временски) и претварање тих узорака у број или низ бројева. Процес дигитализације се може обавити помоћу посебног уређаја или процесора, те процес дигитализације постаје интегрисан са камерама тако да је излаз из камере сигнал у дигиталном облику.

3.1.1 Узимање узорака - узорковање

Просторно узорковање

Дигитална слика може бити генерисана узимањем узорака из аналогног видео сигнала у редовним интервалима. Резултат овог поступка је узоркована слика (енг. *sampled image*) - фрејм (енг. *frame*) дефинисана као низ одабраних тачака таквих да је између сваке две тачке подједнако растојање. Сваки фрејм се може реконструисати представљајући сваки узорак као елемент слике - пиксел. Најчешће је у облику правоугаоника где су одабране тачке тј. узорци позиционирани на квадратној мрежи. Визуелни квалитет фрејма зависи од броја узетих узорака (тачака). Већа резолуција узимања узорака (више узорака) даје финије представљање слике. Међутим већи број узорака захтева већи капацитет за чување података

Временско узорковање

Узимањем узорака видео сигнала у периодичним временским тренуцима добија се серија фрејмова (енг. *moving video image*). Репродукција серије фрејмова проузрокује илузију кретања. Већи број узорака тј. брзина узорковања (енг. *frame rate*) даје глатко појављивање кретања у видео сцени али захтева већи меморијски простор за складиштење. Од 10 до 20 фрејмова по секунди је типично за видео комуникацију малог протока (енг. *low bit rate*), од 25 до 30 фрејмова по секунди је телевизијски стандард док је 50, 60 фрејмова по секунди је прикладно за видео високог квалитета.

Типови узорковања

Видео сигнал може бити узоркован као низ комплетних фрејмова, тзв. прогресивно узорковање (енг. *progressive sampling*) или као секвенца испреплетаних поља (енг. *interlaced sampling*).

У испреплетаној видео секвенци, половина података у фрејму, једно поље (енг. *field*), се узоркује у сваком временском интервалу узорковања. Поље се састоји или од парних или од непарних линија комплетног фрејма. Тако да испреплетана видео секвенца садржи низ поља која садрже половину комплетне информације видео фрејма.

3.1.2 Простор боја

Већини дигиталних видео апликација неопходан је механизам за снимање и представљање информација о боји. Монохроматска слика захтева само један број за означавање бистрине и осветљености сваког пиксела. Друге слике захтевају најмање три броја по пикселу за прецизно представљање боје.

Метод изабран за представљање осветљености и боја назива се простор боја (енг. *colour space*). Најчешће коришћени простори боја за представљање дигиталне слике и видеа су *RGB* и *YCrCb*.

RGB

У простору боја *RGB*, боја сваког пиксела је представљена помоћу три броја које указују на релативне пропорције црвене, зелене и плаве боје. Ове боје представљају основне боје, све друге боје могу се добити њиховом комбиновањем. Како три компоненте имају готово једнак значај у представљању крајње боје, *RGB* системично представља сваку компоненту са истом прецизношћу (истим бројем бита). Уобичајено се користи 8 бита за представљање компоненти, тј. $3 \times 8 = 24$ бита за представљање појединачних пиксела.

Овај простор боја је добро прилагођен за снимање и приказивање слика. Снимање *RGB* слике укључује филтрирање кроз црвене, зелене и плаве компоненте сцене. *Colour Cathode Ray Tubes-CRTs* и *Liquid Crystal Displays-LCDs* приказује *RGB* слику тако што одвојено осветљава црвену, зелену и плаву компоненту сваког пиксела на основу интензитета сваке компоненте. Спајањем ових компоненти добија се “права” боја.

YCrCb

RGB није много ефикасан за представљање боја. Људски визуелни систем (енг. *The human visual system - HVS*) је мање осетљив на боју него на осветљење. У *RGB* простору боја, три боје су подједнако важне и осветљеност је представљена у све три компоненте. Могуће је представити слику ефикасније одвајајући информацију о осветљености од информација о боји.

Често коришћен простор боја овог типа је $Y : Cr : Cb$ (познат још као *YUV*). Y је компонентна која садржи информације о осветљености - лума. Лума се може представити на основу вредности R , G и B :

$$Y = k_r R + k_g G + k_b B$$

где су k коефицијенти.

Информације о боји представљају се посебним компонентама - *chroma*, које представљају разлику између боја и осветљености (*colour chrominance*):

$$C_b = B - Y$$

$$C_r = R - Y$$

$$C_g = G - Y$$

Слика је потпуно одређена компонентом осветљености Y и компонентама боја C_r , C_g , C_b које представљају разлику између интензитета боје и осветљености сваког пиксела.

Збир компоненти боја, $C_b + C_r + C_g$ је константан што омогућава да се само две од три компоненте чувају и преносе, а да се трећа израчунава на основу њих. У простору боја *YCrCb* преносе се само лума(Y) и *chroma*(C_r, C_b).

Предност *YCrCb* у односу на *RGB* огледа се у чињеници да се C_r и C_b компоненте могу представљати са мањом резолуцијом него Y јер је људски визуелни систем

мање осетљив на боје него на осветљење. То смањује количину података потребну за представљање C_r и C_b компоненти без утицаја на квалитет слике.

За обичног посматрача не постоји очигледна разлика између RGB и $YCrCb$ слике са смањеном резолуцијом интезитета боја. $YCrCb$ са смањеном резолуцијом интезитета боја представља једноставан, али ефикасан начин компресије слика.

Глава 4

Компресија видео података

4.1 Компресија података

Компресија података представља начин да се иста информација запише тако да заузима мање меморијског простора. Алгоритми компресије базирају се на чињеници да подаци који се користе садрже много непотребног вишака. По теорији информација тачно се може израчунати колико износи тај вишак и одбацити све оно што је редудантно. Алгоритми компресије се могу поделити у две категорије: алгоритми компресије без губитака и алгоритми компресије са губицима. Компресијом без губитка добија се датотека која је након декомпресије идентична оригиналу. Најпознатији програм за компресију података без губитака је *ZIP*. Због величине ових фајлова почели су да се развијају алгоритми компресије са губицима. Како ови алгоритми дозвољавају губитке у фајлу, декомпресовани фајл није једнак оригиналу. Пре овакве компресије потребно је извршити редукцију података који ће бити пренети и сачувани и то на такав начин да се што мање угрози квалитет и садржај оригиналне информације.

Последњих неколико деценија дошло је до значајних унапређења у области обраде слика и видео сигнала. Нека од ових унапређења односе се управо на компресију слика и видеа.

4.2 Стандарди за видео компресију

Са напретком технологија у области телекомуникација, електронике и компресије стигла је ера дигиталног видеа. Највеће достигнуће у видео компресији представља могућност да се мултимедијална информација која представља компресовану слику, аудио или видео посматра само као још један тип података. То значи да се мултимедијалне информације дигитално кодирају тако да се могу чувати и преносити заједно са другим дигиталним типовима података.

У последњих неколико деценија подаци који представљају слике и видео су толико велики да је неизбежна њихова компресија. Методе компресије засноване су на особинама чула вида које је је осетљивије на енергију са низом просторном фреквенцијом. Из овог разлога слике могу бити компресоване са губитком тако да садрже много мање података него оригиналне слике без значајног губитка квалитета слике.

Развитком техника компресије за пренос и складиштење велике количине података постало је неопходно да се ове технике стандардизују у различитим платформама и апликацијама. Подстакнут је развој апликација и промовисана интероперабилност између система различитих произвођача. Стандардизација је, такође, довела до ра-

звоја исплативих имплементација које промовишу широку употребу мултимедијалних информација [9].

4.2.1 *JPEG*

Од средине 1980-тих година, група *ISO/CCITT* позната као *JPEG (Joint Photographic Experts Group)* почела је рад на креирању ефикасне шеме за кодирање низа слика тј. видеа. Тако је 1992. године креiran *JPEG* стандард (*ISO/IEC 10918*). *JPEG* стандард обезбеђује неколико модела кодирања, од основних до оних софицираних, у зависности од области примене. Данас, *JPEG* представља најчешће коришћен формат у уобичајеном раду са slikama.

4.2.2 *MPEG*

Као одговор на растућу потребу за заједничким форматом за кодирање и складиштење дигиталног видеа, 1988. године, Међународна организација за стандардизацију (*ISO*), основала је *MPEG* групу (*The Moving Picture Expert Group*) са циљем развијања стандарда за компресију аудио и видео садржаја. Прва фаза њиховог рада завршена је 1991. године и представља начин кодирања за дигитално складиштење медија. Овај стандард назива се *ISO 11172* стандард познат још као *MPEG-1* стандард. Друга фаза почела је 1990. године развојем додатака на *MPEG-1* са циљем веће флексибилности и боље отпорности на грешке. Та фаза довела је до развоја *ISO 13818* стандарда. Овај стандард познат је још под називом *MPEG-2*.

MPEG-1 и *MPEG-2* стандарди се користе у комерцијалној производњи. Ови стандарди се баве видео садржајима заснованим на фрејмовима и у многим апликацијама су понудили решење за замену аналогних система који су се користили пре дигиталних. Најважнија улога ових стандарда је чување и ефикасан пренос аудио и видео садржаја.

Са порастом употребе дигиталних медија долази до брисања граница између три некада јасно одвојена сервиса: комуникација, интеракција и емитовање и њима одговарајућих индустриских сектора, пре свега у области телевизије, рачунара и телекомуникација.

Из тог разлога, 1993. године *MPEG* група је покренула нову фазу стандардизације, названу *MPEG-4*. За разлику од *MPEG-1* и *MPEG-2* чији је акценат био на ефикасном кодирању, *MPEG-4* је за циљ имао да стандардизује алгоритме за кодирање садржаја у мултимедијалним апликацијама омогућавајући интерактивност, високу компресију и проширитвост аудио и видео садржаја. Док *MPEG-1* и *MPEG-2* за представљање сцена предлаже парадигму засновану на фрејмовима, *MPEG-4* предлаже парадигму засновану на објектима. Прва верзија овог међународног стандарда пуштена је у пролеће 1999. године.

Постоје још и *MPEG-7* и *MPEG-21* стандарди који не представљају формалне стандарде али заокружују целину. *MPEG-7* је формални систем за опис мултимедијалног садржаја док се *MPEG-21* описује као мултимедијални оквир (енг. *framework*).

4.2.3 ITU-H серије

Паралелно са *ISO MPEG*, *ITU-T* је креирао неколико стандарда за мултимедијалне комуникације, *X.261*, *X.263* и *X.263+* стандарде. *X.261* је стандард дефинисан од стране *ITU-T Study Group XV (SG15)* за примене у телефонији и видео конференцијама. Настао је 1984. године и намењен је за аудиовизуелне услуге протока *bitrate* око $tx384kbit/s$, где је t између 1 и 5. 1988. године је одлучено да се ипак пређе на брзине око $rx64kbit/s$, где је r од 1 до 30. Стандард *X.261* је усвојен у децембру 1990. године. Због своје двосмерне комуникационе природе, одређено је да максимално кашњење у кодирању буде $150ms$. Улазни формати који се користе и који су дефинисани у *X.261* су *CIF – Common Intermediate Format* и *QCIF*.

ITU-T је новембра 1993. године отпочела рад на новом стандарду. Главни циљ је био креирање стандарда погодног за апликације чија је брзина испод $64kbit/s$. На пример, слање видео података преко телефонске мреже јавног сервиса и преко мобилне мреже подразумева брзину од 10 до $24kbit/s$. Током овог пројекта, креирана су два подциља, први, да се побољша *X.261* за апликације мале брзине и други, дизајнирање стандарда који се у основи разликује од *X.261*, а који ће допринети бољем квалитету. Тако су, као резултат првог подциља настали *X.263* и *X.263+* стандарди, а као резултат другог *X.261L* за који се очекивало да ће се ускладити са *MPEG-4* стандардом.

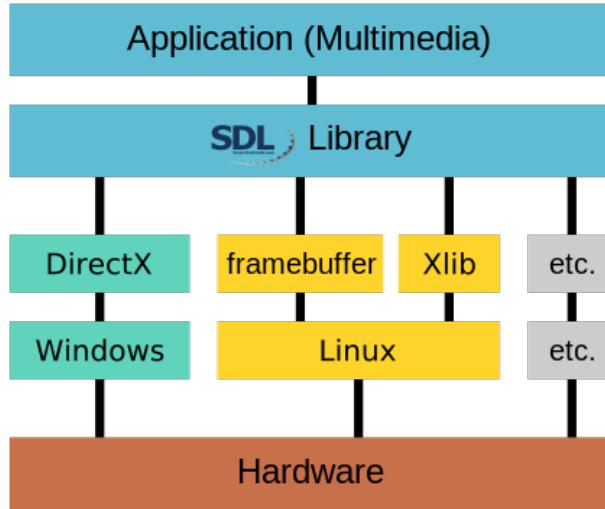
Глава 5

Библиотека *SDL*

SDL (*Simple DirectMedia Layer*) – је развојна библиотека дизајнирана да обезбеди слој ниског нивоа хардверске апстракције за рачунарске мултимедијалне компоненте. Обезбеђује приступ аудио и видео садржају, тастатури, мишу, цојсттику и графичком окружењу преко *OpenGL* и *Direct3D-a*. Користи се у развоју игара, емулатора, итд [10].

Библиотека је бесплатна за коришћење, *SDL 2.0* и новије верзије *SDL-a* су дистрибуиране под *zlib* лиценцом док су старије верзије (*SDL 1.2* и старије) доступне под *GNU LGPL* лиценцом. Библиотека је креирана 1998. године од стране инжењера *Sam Lanting-a*.

Библиотека званично подржава *Windows*, *Mac OS*, *Linux*, *IOS* и *Android* оперативне системе. Написана је у језику *C*. Такође, постоји могућност повезивања *SDL-a* са још неколико других језика попут *Python-a*.



Слика 5.1: *SDL* слојеви апстракције

На слици су приказани слојеви апстракције “изнад” и “испод” *SDL* библиотеке. Најнижи слој апстракције, односно директан приступ хардверу је “сакривен” - *SDL* библиотека има врло интуитиван и једноставан апликациони програмски интерфејс (*API, Application programming interface*) чиме се смањује комплексност програмирања. Међутим, уколико *SDL* не остварује жељене резултате, програмер не може ни на који начин да утиче на то.

У овом раду је коришћена *SDL 1.2* верзија као програмска подршка за репродукцију видео садржаја. Више о томе биће у поглављу - Имплементација.

Глава 6

Библиотека *OpenCV*

OpenCV је библиотека програмских функција усмерена ка машинском учењу и анализи, модификацији и обради слика на високом нивоу (*computer vision*). Креирана је како би обезбедила заједничку инфраструктуру за апликације које обрађују слике и убрзао сам процес обраде слика [11].

Садржи преко 2500 оптимизованих алгоритама, који укључују како класичне тако и савремене рачунарске алгоритме у области обраде слика. Ови алгоритми се могу користити за детекцију и препознавање лица, идентификацију предмета, праћење покретних објеката, повезивање слика у једну слику високе резолуције (*stitching*), уклањање црвених очију са фотографија снимљених помоћу блица итд.

OpenCV је објављена под *BSD* лиценцом, те је бесплатна како за академску тако и за комерцијалну употребу. Библиотека је написана у језику *C++* и њен главни интерфејс је у *C++*, такође има и *C*, *Python*, *Java* и *MATLAB* интерфејс. Подржава *Windows*, *Linux*, *Mac OS*, *iOS* и *Android* оперативне системе. Дизајнирана је са строгим фокусом на апликације у реалном времену (*realtime*).

У овом раду функције ове библиотеке су коришћене за трансформацију и рад са појединачним фрејмовима видеа.

Глава 7

Библиотека *FFmpeg*

FFmpeg је водећи развојни оквир (енг. *Framework*) за кодирање, декодирање, филтрирање и друго рукување аудио и видео садржајем. Пројекат креирања *FFmpeg-a* започео је Фабрис Белард (*Fabrice Bellard*) 2000. године, а од 2004. године вођа пројекта је Михаел Нидермајер (*Michael Niedermayer*). Пројекат *FFmpeg* тежи да обезбеди најбоље технички могуће решење како за програмере тако и за крајње кориснике апликације [12]. *FFmpeg* чини скуп библиотека:

- *libavcodec* - функције за кодирање и декодирање аудио и видео садржаја применом великог броја стандарда за компресију
- *libavutil* - функције које поједностављују сам процес програмирања аудио/видео система, укључујући генераторе случајних бројева, функције за решавање различитих математичких проблема, услужне мултимедијалне функције, итд
- *libavformat* - функције за учитавање аудио/видео садржаја (демултиплексирање) и његово уписивање(мултиплексирање)
- *libavfilter* - функције за филтрирање садржаја
- *libswscale* - функције које омогућавају високо оптимизовано скалирање слика
- *libswresample* - функције које омогућавају конверзије између формата

FFmpeg нуди висок ниво апстракције који омогућава успешно рукување аудио/видео садржајем без потребе за спуштањем на ниво регистрара. Доступне функције и структуре су веома добро документоване, што употребу *FFmpeg-a* чини изузетно једноставном.

FFmpeg је написан већином у програмском језику С и мањим делом у асемблеру. Иако је писан у процедуралном језику организован је водећи се идејама објектно оријентисане парадигме. Структурама података представљене су логичке целине којима је или описан аудио/видео садржај (нпр. контејнер датотека, пакет, елементарни ток,...) или се користе за обраду над њим (нпр. декодер, енкодер, ...). Структуре су организоване тако да поред поља која носе све потребне информације о посматраној целини постоје и показивачи на функције којима се врши одређена обрада аудио/видео садржаја.

Примењује се концепт функција омотача (енг. *wrapper function*), тј никада се директно не позивају функције на које указују поља из структура података, већ постоје функције омотачи које као један од аргумента примају показивач на одговарајућу структуру података. Након иницијализације и провере коректности одређених поља

те структуре, омотач функција позива функцију на коју указује одговарајуће поље структуре. Затим та функција врши неку обраду садржаја. На коју ће конкретно функцију показивати поље из одговарајуће структуре зависи од типа аудио/видео садржаја над којим се врши обрада. На пример, структура којом је представљена улазна контејнер датотека, има између осталог, функције за читање заглавља и за читање појединачних пакета из контејнер датотеке. *FFmpeg* у својој “бази” функција, поседује функције којима се читају заглавља, односно пакети из улазне контејнер датотеке за велики број формата датотеке. У зависности од формата приликом иницијализације структуре, која представља улазну контејнер датотеку, показивач на функцију за читање заглавља, односно пакета, ће бити постављен на функцију која одговара специфицираном типу улазне датотеке.

FFmpeg пројекту се стално додају нове могућности. Отприлике се на свака три месеца прави ново велико издање које садржи исправке грешака из претходних издања.

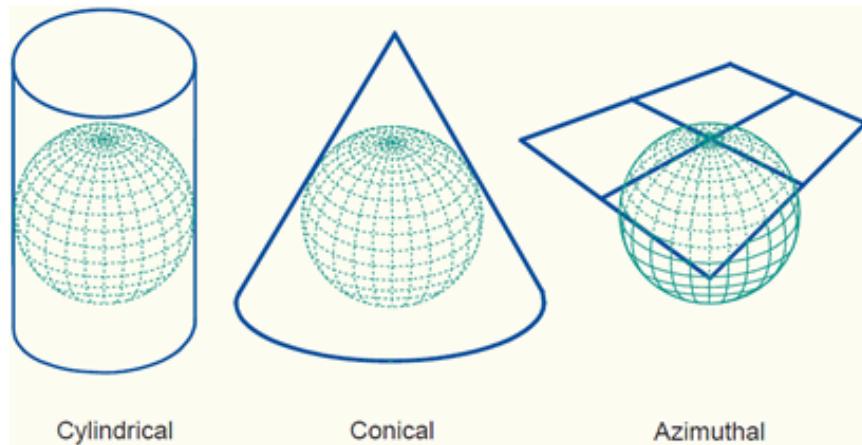
Глава 8

Фрејмови видеа и трансформације

8.1 Еквидистантна цилиндрична пројекција

Перспективна пројекција, по дефиницији, подразумева премештање тачака, које се порсматрају из неке фиксне тачке простора, на површину. Под пројектовањем се подразумевају одређена правила која су везана за спољашњи изглед објекта, углове и многе друге карактеристике. Приликом пројекције сфере на раван потребно је истаћи којом површи ће та сфера бити обухваћена. Постоје три начина да се обухвати сфера, то су:

- Цилиндрична пројекција
- Конусна пројекција
- Зенитна пројекција



Слика 8.1: Цилиндрична, конусна и зенитна пројекција

У овом раду од значаја нам је цилиндрична пројекција и то она пројекција код које су одређене дужине једнаке, тзв. еквидистантна цилиндрична пројекција (енг. *Spherical projection, Equirectangular projection*). Имплементирани видео плејер сферичног видеа као улазни параметар користи видео чији су фрејмови у облику еквидистантне цилиндричне пројекције. Ова пројекција је облик цилиндричне пројекције која приказује 360° хоризонталног и 180° вертикалног видног поља. Видео садржај чији су фрејмови у облику ове пројекције је специјално креиран и намењен за приказивање помоћу видео плејера сферичног видеа. Креиран је помоћу специјализованог система од

неколико камера које заједно покривају и снимају све углове сцене, 360 x 180. Снимци који се направе помоћу тог система камера се затим повезују у један видео садржај (енг. *video stitching*). Овај процес подразумева комбиновање различитих видеа (чији се садржај преклапа) у циљу стварања једног видеа високе резолуције. Обавља се помоћу специјализованог рачунарског софтвера. Неки од ових софтвера су бесплатни и доступни на интернету [6][7].

Видео плејер је имплементиран тако да се из улазног видеа читају фрејмови који се трансформишу и тако трансформисани приказују на екрану. Превлачењем миша се задају вредности које одређују који део таквог фрејма се приказује на екрану. На основу тих вредности врши се трансформација фрејмова. Ове трансформације за право представљају ротације сфере која је представљена еквидистантном цилиндричном пројекцијом. Поред ових трансформација, врши се и трансформација фрејма из еквидистантне цилиндричне пројекције у правоугаону пројекцију и фрејм се у таквом облику приказује на екран. За трансформације фрејмова користи се библиотека *OpenCV*.

8.2 *OpenCV* и функција *remap()*

Основна класа за представљање и складиштење слика у библиотеци *OpenCV* је класа *Mat*. *Mat* се састоји из два дела: заглавља, које садржи информације о димензијама слике и адреси на којој се слика складишти, и показивача на матрицу која садржи вредности пиксела који чине слику. Стога се свака слика висине m и ширине n може замислити као матрица $m \times n$. Матрица се у програмирању представља помоћу низа. Трансформација фрејмова имплементирана је помоћу *OpenCV* библиотеке, пре свега помоћу функције *remap()* ове библиотеке.

```
void remap (
    InputArray src,
    OutputArray dst,
    InputArray map1,
    InputArray map2,
    int interpolation,
    int borderMode,
    const Scalar& borderValue
)
```

Ова функција примењује генеричку геометријску трансформацију на слику. Омогућава мапирање пиксела слике, односно њихово премаштање из једне позиције на другу. Први параметар, *src*, представља почетну, улазну слику, док други параметар, *dst* представља, излазну слику, насталу као резултат примене ове функције. Други и трећи параметар представљају матрице (имплементиране помоћу низова) које одређују како се мапирају пиксели, односно који пиксел резултујуће слике је заправо мапирани пиксел почетне слике. Мапирање се врши у складу са следећом формулом:

$$dst(x, y) = src(mapX(x, y), mapY(x, y))$$

Пиксел резултујуће слике записан у врсти x и колони y има вредност једнаку вредности пиксела који је записан у врсти $mapX(x, y)$ и $mapY(x, y)$ почетне слике. Функција $remap()$ се може користити за креирање различитих ефеката, попут ефекта сочива, обртања слика као у огледалу и многих других.

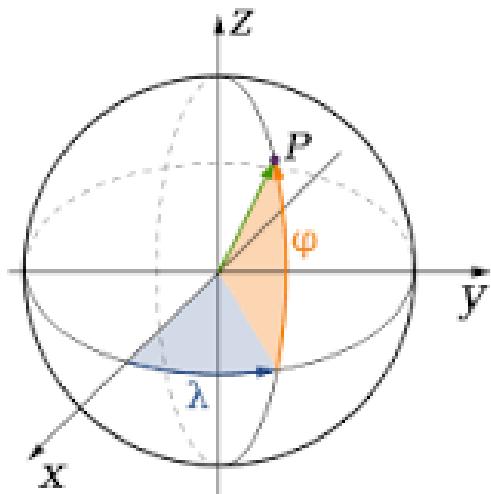
8.3 Трансформације фрејмова, *cubemap* пројекција

Фрејмови представљају еквидистантну цилиндричну пројекцију и покривају 360×180 видног поља. На слици .. представљен је један улазни фрејм видеа који је преузет са .



Слика 8.2: Улазни фрејм видеа

Фрејмови видеа покривају 360×180 видног поља и представљају неку врсту панораме. Панораме у облику ове пројекције изгледају као да је слика умотана у сферу и да се затим та слика посматра из центра сфере.



Слика 8.3: Сфера у координатном систему

Видео плејер је имплементиран тако да користи трансформације које на основу еквидистантне цилиндричне пројекције креирају пројекцију у облику коцке (енг. *cubemap*). Пројекција у облику коцке заправо представља други начин за представљање неког окружења, помоћу комбинације шест страница коцке где свака страница носи одговарајући садржај. Ова техника се већ дugo користи у рачунарској графици. На основу фрејма који представља еквидистантну цилиндричну пројекцију могуће је креирати шест нових фрејмова.

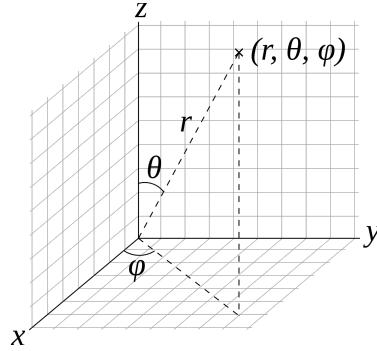


Слика 8.4: Пројекција у облику коцке, шест страница коцке

На овај начин добијају се фрејмови који представљају кадрове који би се видели уколико би се из центра сфере гледало напред, назад, лево, десно, горе, доле. Као излаз на екрану приказује се само један фрејм од ових шест. Из тог разлога плејер је имплементиран тако да се на основу вредности задатих помоћу миша, прво изврше трансформације које представљају ротацију сфере око y и z осе. Након тих трансформација фрејм остаје у облику еквидистантне цилиндричне пројекције. Затим се од таквог новокреираног фрејма креира фрејм који представља једну страницу коцке ("предња страница"). Такав фрејм се онда приказује на екрану.

8.3.1 Креирање странице коцке

За представљање тела у тродимензионалном простору уобичајено се користи правоугли координатни систем са координатама x , y , z , познат још и као картезијанов координатни систем (енг. *Cartesian*). Поред овог координатног система, користи се и сферни координатни систем, са координатама (r, θ, ϕ) . Прва координата, r , представља удаљеност од центра координатног система, θ угао који заклапа права која спаја тачку са координатним почетком са позитивним делом z осе, док ϕ представља угао који права која спаја тачку са координатним почетком заклапа са позитивним делом x осе.



Слика 8.5: Веза између правоуглог и сферног координатног система

Правоугле координате x, y, z , се на основу сферних координата добијају следећим формулама:

$$x = r \sin \theta \cos \phi \quad (8.1)$$

$$y = r \sin \theta \sin \phi \quad (8.2)$$

$$z = r \cos \theta \quad (8.3)$$

Замислимо имагинарну сферу која је пројектована еквидистантном цилиндричном пројекцијом. Уколико је у ту сферу уписана коцка, чија је дужина странице два и уколико је центар сфере (уједно и коцке) постављен у нулту тачку правоуглог координатног система, тада странице коцке припадају равнима $x = 1, x = -1, y = 1, y = -1, z = 1, z = -1$. Посматрајмо страницу коцке на равни $x = 1$. Из (8.1) се r може изразити као:

$$r = \frac{1}{\sin \theta \cos \phi}. \quad (8.4)$$

Из (8.2) и (8.4) може се изразити y :

$$y = \tan \phi. \quad (8.5)$$

Из (8.3) и (8.4) може се изразити z :

$$z = \frac{\cot \theta}{\cos \phi}. \quad (8.6)$$

Дакле, правоугле координате странице коцке, која лежи у равни $x = 1$, изражене помоћу сферних координата:

$$(1, \tan \phi, \frac{\cot \theta}{\cos \phi}) \quad (8.7)$$

Уколико се даље изразе сферне координате помоћу правоуглих, из (8.5), се добија:

$$\phi = \arctan y, \quad (8.8)$$

док се из (8.7) добија

$$\theta = \arctan(z \cos \phi) \quad (8.9)$$

Фрејмови у облику еквидистнатне цилиндричне пројекције се трансформишу у фрејмове који представљају страницу коцке која лежи у равни $x = 1$. Ове трансформације засноване су на поменутим везама између координатних система. Заправо се, помоћу поменутих формулa, израчунају θ и ϕ углови и матрице $tarX$ и $tarY$ се попуњавају овим вредностима. На тај начин се са еквидистантне цилиндричне пројекције, која представља сферу, прелази на пројекцију једне странице коцке, која је уписана у ту сферу.



Слика 8.6: Еквидистантна цилиндрична пројекција и пројекција у облику једне странице коцке

8.3.2 Трансформације фрејмова, ротација сфере

Од фрејма у облику еквидистантне цилиндричне пројекције на екрану се не приказује читав фрејм. Приказује се само део тог фрејма и то онај сачињен од пиксела који учествују у креирању "предње" странице коцке. Као што је већ напоменуто, превлачењем миша се задају вредности које одређују који део фрејма се приказује на екрану. Заправо, кориснику се омогућава да изабере правца и промени угао посматрања. На овај начин се одређује који део кадра ће бити приказан. Метод за креирање предње странице, односно правоугаоне пројекције, увек мапира исте пикселе. Како би се променио угао који се приказује неопходно је пре креирање правоугаоне пројекције трансформисати фрејм. Фрејм се трансформише тако што се над њим врше трансформације које заправо представљају ротацију сфере представљене еквидистантном цилиндричном пројекцијом. Ротација се врши око y и z осе.

Дакле, уколико корисник померањем миша "захтева" промену у десно за угао α , и на горе за угао β . Плејер ће прво извршити трансформације које представљају ротације сфере, и то тако што се прво изврши ротација око z осе за угао α , а затим ротација око y осе за угао β . Како је фрејм у облику цилиндричне пројекције, "десни крај" пројекције се може надовезати на "леви", те није неопходно вршити ротацију сфере око z осе, већ је за померање у лево или у десно довољно извршити трансформацију која транслира пикселе по хоризонталној оси. Како су формуле за транслацију нешто једноставније него формуле за ротацију око осе, на овај начин добија се на брзини израчунавања.



Слика 8.7: Почетни и трансформисани фрејмови

Трансформација фрејма за померање кадра горе/дело врши се помоћу матрице ротације сфере око y осе за угао θ .

$$My(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Ова трансформација одвија се у три корака. Први корак подразумева мапирање пиксела тако да се са слике односно фрејма који је у дводимензионом простору пређе на тродимензионалне координате. То подразумева да се за сваки пиксел фрејма израчуна угао у односу на центар како хоризонтално тако и вертикално. Затим се трансформишу сферичне координате у правоугле координате x, y, z (правоугаоне координате). Други корак обухвата примену формула за ротацију сфере у тродимензионалном простору помоћу поменуте матрице ротације.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = My(\theta) * \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Трећи корак подразумева трансформацију којом се картезијанове координате враћају назад у сферичне. На овај начин креира се фрејм који представља еквидистантну цилиндричну пројекцију заротиране сфере.



Слика 8.8: Почетни и трансформисани фрејмови, ротација сфере око y осе

Глава 9

Имплементација видео плејера

Ток (енг. *stream*) представља низ података доступних у временену. У имплементацији видео плејера сферичног видеа обрађују се само видео токови. Сваки стрим је кодиран од стране различитих врста кодека (енг. *codec*). Кодек дефинише како су стварни подаци кодирани и декодирани (енг. *COded and DEEncoded* – одатле назив *CODEC*). Делови тока се представљају пакетима. Пакети могу да садрже битове података који се декодирају у фрејмове. Фрејмови се затим обрађују и приказују на екрану. Сваки пакет садржи комплетан фрејм или више фрејмова у случају аудија.

Апликација у којој је имплементиран видео плејер сферичног видеа написана је у програмском језику *C++*, коришћењем библиотека *FFmpeg* и *OpenCV*. Приступ хардверским компонентама врши се уз помоћ библиотеке *SDL*. Реализована је тако да се њено покретање врши прослеђивањем адекватног улазног параметра тј. видеа који се репродукује, у виду аргумента командне линије. Улазни параметри могу бити видео датотеке великог броја формата.

Саму срж видео плејера сферичног видеа представља класа *Player*. Ова класа је задужена за отварање видео тока, декодирање, рад са аудио пакетима и репродуковање видео садржаја. Поред ове класе, креирана је и класа *Button* која је задужена за руковање догађајима проузрокованих помоћу миша.

9.1 Иницијализација и учитавање улазног тока

У главној нити (енг. *main*), прво се креира објекат класе *Player* и позива се метод *init()* ове класе. Овај метод иницијализује све доступне видео формате и кодеке који су подржани од стране *FFmpeg-a* помоћу функције:

```
void av_register_all()
```

Осим тога, врши се иницијализација *SDL* библиотеке. Библиотека мора бити иницијализована пре позивања било које *SDL* функције. Након иницијализације, врши се учитавање датотеке позивом методе *openFile()* класе *Player* која позива функцију *FFmpeg* библиотеке:

```
int avformat_open_input(
    AVFormatContext **ps ,
    const char *filename ,
    AVInputFormat *fmt ,
    AVDictinary **options
)
```

Ова функција као параметар узима путању до улазне датотеке (која се прослеђује као аргумент команде линије), отвара је и чита њено заглавље.

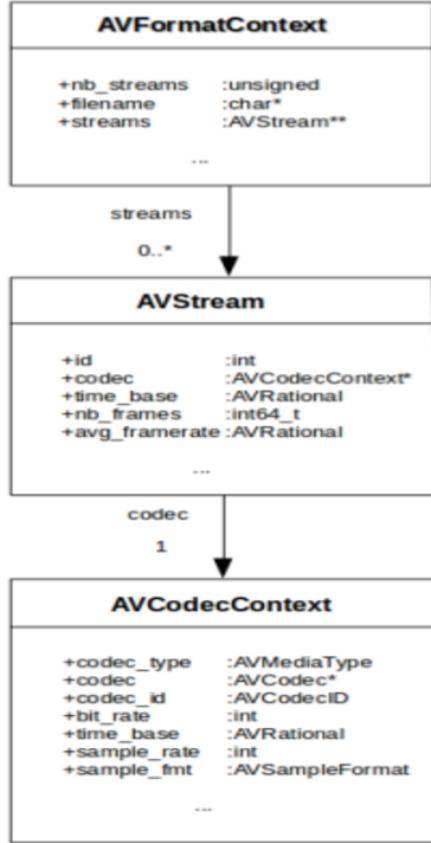
Позивом функције *avformat find stream info()* читају се пакети из елементарног тока све док се не одреде потребне информације о компресованом садржају који се налази у улазној датотеци. Такође, ова функција рачуна брзину протока (енг. *framerate*).

Функције *avformat open input()* и *avformat find stream info()*, на основу података из улазне датотеке, попуњавају поља структуре *AVFormatContext*.

Структуром *AVFormatContext* се у потпуности описује аудио/видео садржај и она представља полазну тачку за сваку даљу обраду. Структура садржи показиваче на две структуре *AVInputFormat* и *AVOutputFormat*. У зависности од типа датотеке чији се садржај представља, иницијализује се једна од наведених структура (за улазну датотеку *AVInputFormat*, за излазну датотеку структура *AVOutputFormat*). Функцијама *avformat open input()* и *avformat find stream info()* се иницијализују поља структуре *AVFormatContext* за потребе представљања улазне датотеке, а самим тим и структура *AVInputFormat*.

Структура је значајна због показивача на функције којима се чита садржај из улазне датотеке (читање заглавља, читање пакета из елементарних токова, итд.) Функције за читање садржаја се разликују у зависности од формата улазне датотеке. У својој бази *FFmpeg* поседује функције за читање садржаја на основу правила за велики број различитих формата. На основу улазне датотеке, у структури *AVInputFormat* се постављају показивачи на одговарајуће функције за читање. Поред основних информација о аудио/видео садржају и показивача на функције за читање садржаја у структури *AVFormatContext* се налази и низ показивача на елементарне токове података (енг. *elementary stream*), који су представљени структуром *AVStream*. Структура *AVStream* носи информације о једном елементарном току података.

На слици је приказан дијаграм класа са основним структурима које се користе за представљање видео садржаја као и њихова међусобна повезаност.



Слика 9.1: Дијаграм класа релевантних структура FFmpeg библиотеке

AVStream структура у својим пољима садржи поље које носи информације о кодеку за тај елементарни ток података, садржане у структури *AVCodecContext* која је такође представљена дијаграмом. На основу поља *codec type* се испитује ког је типа елементарни ток. Елементарни ток података може да носи аудио и видео информација, али такође може да носи и текстуалне податке, превод, итд.

У овом раду биће разматрана обрада елементарних токова који су видео типа. Наиме, након што се из низа показивача на елементарне токове издвоје они који указују на видео садржај, врши се даља обрада видео сигнала, која ће бити описана у наставку.

9.2 Иницијализација кодека

Након што се у главној нити изврше метод *init()* и метод *openFile()* класе плејер позива се метод *initCodec()* који врши иницијализацију кодека. Наиме, након што се утврди да елементарни ток података који треба обрадити садржи видео садржај, унутар структуре *AVCodecContext* се чувају информације које у потпуности описују на који начин је видео кодиран, како би декодирање садржаја било ефикасно.

На основу јединственог идентификатора кодека који је садржан унутар поменуте структуре, алоцира се меморија и иницијализују се подразумеване вредности струк-

туре које носе информације о конкретном кодеку везаном за тај елементарни ток података.

```
bool Player::initCodec() {
    int videoStream = -1;

    for (unsigned i = 0; i < m_formatCtx->nb_streams; i++) {
        if (m_formatCtx->streams[i]->codec->codec_type == AVMEDIA_TYPE_VIDEO) {
            videoStream = i;
            break;
        }
    }

    if (videoStream == -1) {
        std::cout << "Didn't find a video stream!" << std::endl;
        return false;
    }

    m_codecCtxOrig = m_formatCtx->streams[videoStream]->codec;
    m_codec = avcodec_find_decoder(m_codecCtxOrig->codec_id);

    if (m_codec == NULL) {
        std::cout << "Unsupported codec!" << std::endl;
        return false;
    }
}
```

9.3 *SDL* имплементација

У имплементацији видео плејера библиотека *SDL* коришћена је за обраду догађаја проузрокованих помоћу миша као и за репродуковање излазног садржаја односно приказивање тог садржаја на екран.

9.3.1 *SDL* нит

Како се ослушкивање и обрада догађаја коришћењем библиотеке *SDL* одвија и мора одвијати у главној нити, креира се помоћна *SDL* нит која је задужена за репродуковање видео садржаја. Овој *SDL* нити прослеђује се као аргумент објекат класе *Player* и она над тим објектом позива метод *play()* који представља основу читаве апликације.

```

typedef struct {
    Player *player;
} ThreadData;

int main( int argc , char* args [] ) {

    SDL_Thread *thread = NULL;
    ThreadData *data;

    ...
    data = (ThreadData *) malloc( sizeof(ThreadData) );
    data->player = player;

    thread = SDL_CreateThread( playVideo , data );
    ...
}

int playVideo ( void *data )
{
    ThreadData *tdata = (ThreadData *) data;
    Player * player = tdata->player;

    player->play();
    ...
}

```

Метод *play()* имплементиран је тако да се пролази кроз улазни ток, издвајају они пакети који носе видео садржај, декодирају видео фрејмови на основу тих пакета и врши трансформација фрејмова из еквидистантне цилиндричне пројекције у правоугаону пројекцију. Затим се трансформисани фрејмови приказују на екран.

За пролазак кроз улазни ток података користи се функција *FFmpeg* библиотеке:

```
int av_read_frame(AVFormatContext *s , AVPacket *pkt).
```

Ова функција даје следећи фрејм тока и чува га у структури *AVPacket*, задатој као други аргумент функције. Не проверава да ли је фрејм валидан за декодирање. Декодирање видео фрејмова на основу тих пакета врши се помоћу функције:

```

int avcodec_decode_video2(
    AVCodecContext *avctx ,
    AVFrame *picture ,
    int *got_picture_ptr ,
    const AVPacket *avpkt
).

```

Ова функција декодира фрејм на основу вредности из *AVPacket*, дате као четврти аргумент, и чува га у *AVFrame *picture*. Затим се врши трансформација фрејмова чији је поступак описан у претходном поглављу. За приказивање фрејмова на екрану такође је коришћена *SDL* подршка.

9.3.2 *SDL* и исцртавање фрејмова

SDL има неколико метода за исцртавање слика на екран. Посебно је погодан за тзв. *YUV Overlay*, односно приказивање филмова (видео садржаја) на екрану. *YUV* заправо представља *YCrCb*.

Основна област за приказивање слика коришћењем *SDL-a* је површина (енг. *surface*). Ова површина креира се помоћу функције

```
SDL_Surface *SDL_SetVideoMode(  
    int width,  
    int height,  
    int bitsperpixel,  
    Uint32 flags)
```

Након креирања површине на њој се креира *YUV Overlay* за приказивање видеа:

```
SDL_Overlay *SDL_CreateYUVOverlay(  
    int width,  
    int height,  
    Uint32 format,  
    SDL_Surface *display);
```

Затим се помоћу функције *sws scale()* трансформисани фрејм конвертује у *YUV* формат и такав записује у структуру *AVPicture*. Показивачи *data* и *linesize* те структуре се постављају тако да показују на поља креiranог *YUV Overlay-a*. Такође се креира *SDL Rect* који одређује где на екрану ће се приказивати видео.

Даље преостаје приказати видео "лепљењем" *YUV Overlay* на површину позивом функције:

```
SDL_Surface *SDL_SetVideoMode(  
    int width,  
    int height,  
    int bitsperpixel,  
    Uint32 flags).
```

Имплементација методе *play()* класе *Player* која представља срж плејера:

```
void Player :: play( int videoStream ) {  
  
    AVPacket         packet ;  
    SDL_Overlay     *bmp ;  
    SDL_Rect        rect ;  
  
    int frameFinished ;  
  
    bmp = SDL_CreateYUVOverlay (   
        m_codecCtx->width ,  
        m_codecCtx->height ,  
        SDL_YV12_OVERLAY ,  
        m_screen ) ;  
  
    while ( av_read_frame ( m_formatCtx , &packet ) >= 0 && ( m_playQuit == false ) ) {  
  
        // Is this a packet from the video stream ?  
        if ( packet . stream_index == videoStream ) {  
  
            avcodec_decode_video2 ( m_codecCtx , m_frame , &frameFinished , &packet ) ;  
  
            if ( frameFinished ) {  
                if ( ! nextFrameNotShow ) {  
  
                    sws_scale (   
                        m_sws_ctxToRGB ,  
                        ( uint8_t const * const * )( m_frame->data ) ,  
                        m_frame->linesize ,  
                        0 ,  
                        m_codecCtx->height ,  
                        m_frameRGB->data ,  
                        m_frameRGB->linesize  
                    ) ;  
  
                    transformFrame ( m_codecCtx->width , m_codecCtx->height ) ;  
  
                    // Display image  
                    SDL_LockYUVOverlay ( bmp ) ;  
                    AVFrame pict ;  
  
                    pict . data [ 0 ] = bmp->pixels [ 0 ] ;  
                    pict . data [ 1 ] = bmp->pixels [ 2 ] ;  
                    pict . data [ 2 ] = bmp->pixels [ 1 ] ;  
  
                    pict . linesize [ 0 ] = bmp->pitches [ 0 ] ;  
                    pict . linesize [ 1 ] = bmp->pitches [ 2 ] ;  
                    pict . linesize [ 2 ] = bmp->pitches [ 1 ] ;  
  
                    // Convert the image into YUV format that SDL uses  
                    sws_scale (   
                        m_sws_ctxToYUV ,
```

```
        ( uint8_t const * const * )
                ( m_frameConvert->data ) ,
        m_frameConvert->linesize ,
                0 ,
        m_codecCtx->height ,
        pict.data ,
        pict.linesize
    );
SDL_UnlockYUVOverlay( bmp );

rect.x = 0;
rect.y = 0;
rect.w = m_codecCtx->width;
rect.h = m_codecCtx->height;

nextFrameNotShow = false;
SDL_DisplayYUVOverlay( bmp, &rect );
}
}
// Free the packet that was allocated by av_read_frame
av_packet_unref(&packet);
}
}
```

9.3.3 *SDL* ослушкивање и обрада догађаја

Како је већ поменуто ослушакивање и обрада догађаја (енг. *handle event*) одвија се у главној нити програма. Прво се креира објекат класе *Button*. Затим се ослушају да ли се неки догађај десио позивом функције:

```
int SDL_PollEvent(SDL_Event *event).
```

Уколико се ухвати (енг. *catch*) неки догађај врши се његова даља обрада коришћењем метода *handleEvents()* класе *Button*. Наиме реагује се само на догађаје проузроковане помоћу миша и то на оне настале превлачење мишем (енг. *drag and drop*) као и на притисак на дугме за затварање прозора (дугме *x*).

```
int main( int argc , char* args [] )
{
    ...
    Player *player = new Player ();
    ...

    Button *myButton = new Button( player );

    while ( ! player->isActive () )
    {
        //If there 's events to handle
        if ( SDL_PollEvent(&event) )
        {
            //Handle button events
            myButton->handleEvents( event );
        }
    }
    ...
}
```

На притисак на дугме за затварање прозора реагује се постављањем вредности поља *active* креираног објекта класе *Player* на вредност која прекида рад помоћне нити. Помоћна нит зауставља приказивање видеа и апликација се затим гаси. На догађаје настале превлачењем миша такође се реагује променом вредности поља објекта класе *Player* на основу којих се врши трансформација фрејмова.

Библиографија

- [1] Virtual Reality 101. 2017. URL <https://www.cnet.com/special-reports/vr101/>.
- [2] 360Provideo. 2017. URL <http://360provideo.hr/>.
- [3] Virtual Reality Society 2017. URL <https://www.vrs.org.uk/virtual-reality/who-coined-the-term-virtual-reality>.
- [4] VR Svet. 2017. URL <http://vrsvet.com/>.
- [5] Virtualna okruženja - Interaktivna 3D grafika i njene primjene. Igor S. Pandžić, Tomislav Pejsa, Kresimir Matković, Hrvoje Benko, Aleksandra Čereković, Maja Matijašević. 2011.
- [6] The Top Video Stitching Software For Perfect 360 Degree View. 2017. URL <https://filmora.wondershare.com/video-editing-tips/video-stitching-software.html>.
- [7] Stitch and create 360 videos automatically 2017. URL <http://www.kolor.com/autopano-video/>.
- [8] Video Codec Design - Developing Image and Video Compression System Iain E. G. Richardson The Robert Gordon University, Aberdeen, UK
- [9] Design of Digital Video Coding Systems – Signal Processing and Communications. Jie Chen, UtVa Koc, K.J.Ray Liu. Marcel Dekker, inc. New York - Basel, 2002.
- [10] SDL - Simple DirectMedia Layer. URL <https://www.libsdl.org/>.
- [11] OpenCV library. URL <http://opencv.org/>.
- [12] FFmpeg. URL <https://www.ffmpeg.org/>.