VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GUI KE STAVBĚ GRAFU FILTRŮ PRO FFMPEG

GUI TO FFMPEG FOR BUILDING A FILTER GRAPH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ROMAN SICHKARUK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. DAVID BAŘINA, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19489/2016/xsichk00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Sichkaruk Roman**
Obor: Informační technologie
Téma: **GUI ke stavbě grafu filtrů pro FFmpeg**
GUI to FFmpeg for Building a Filter Graph
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s multimediálními frameworky. Zaměřte se na zpracování obrazu, grafy filtrů a framework FFmpeg. Vyzkoušejte si tvorbu grafu filtrů nad videem.
2. Navrhněte architekturu grafického rozhraní, které umožní jednoduchou tvorbu grafů pomocí myši. Rozhraní musí umožnit zobrazení náhledu filtrovaného videa.
3. Implementujte navržené rozhraní. Výsledná aplikace musí dovolit zadávat filtrům parametry.
4. Srovnejte navrženou aplikaci s podobnými nástroji pro jiné multimediální frameworky (GStreamer, DirectShow). Diskutujte omezení vaší aplikace a navrhněte směry dalšího pokračování práce.

Literatura:

- dokumentace FFmpeg (http://www.ffmpeg.org/documentation.html)

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2, rozpracovaná aplikace.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bařina David, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá tvorbou grafického uživatelského rozhraní pro tvorbu grafů filtrů FFmpeg. Obsahuje úvod do problematiky, návrh vzhledu, struktury, popis implementace této aplikace a srovnání s již existujícími nástroji pro jiné multimediální frameworky. Výsledná aplikace umožňuje vytvářet a upravovat grafy video filtrů, upravovat parametry jednotlivých filtrů a přehrávat náhled upraveného videa.

Abstract

This thesis describes creation of graphical user interface for FFmpeg filter graph conceiving tool. It includes introduction to problematics, design of an application look, structure, its implementation and comparison with similiar existing tools for another frameworks. Final tool allows user to create and edit video filter graphs, edit parameters of each filter and preview edited video.

Klíčová slova

video filtr, FFmpeg, graf filtrů, GUI

Keywords

video filter, FFmpeg, filter graph, GUI

Citace

SICHKARUK, Roman. *GUI ke stavbě grafu filtrů pro FFmpeg.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bařina David.

GUI ke stavbě grafu filtrů pro FFmpeg

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Roman Sichkaruk
16. května 2017

Poděkování

Děkuji vedoucímu práce panu Ing. Davidu Bařinovi za odborné rady a trpělivost.

Obsah

Úvod

V dnešní době se s pojmem filtr setkáváme docela často. Ať už jsou to obrazové filtry pro úpravu fotografií nebo filtry měnící zvukové signály (aplikace pro změnu hlasu) nebo filtry videa poskytující například zlepšení kvality, odstranění šumu, zrychlení, zpomalení atd. Proto není divu, že vznikají rozhraní, která mají zjednodušit použití těchto filtrů v daných aplikacích. Rozhraní, která zapouzdřují práci s multimédii a mimo jiné umožňují pracovat s filtry se označují jako multimediální frameworky. Zapouzdřují před programátorem docela velkou část pro něj nepodstatné implementace. Tím se šetří čas a úsilí. Místo nutnosti znát, jak filtr naprogramovat, stačí pouze vědět, jak jej použít nebo například sloučit s jinými filtry – vytvářet sekvence filtrů (tzv. grafy filtrů). Tyto grafy mají většinou přesnou syntaxi, moderní frameworky ovšem nabízí nástroje, které mají intuitivní grafické uživatelské rozhraní pro jejich tvorbu. Právě vývoj nástroje pro tvorbu grafů filtrů FFmpeg je tématem této práce. Jelikož multimediální framework FFmpeg takový nástroj postrádá, může být tato bakalářská práce reálně využita ke studijním účelům nebo pro zjednodušení práce s frameworkem FFmpeg.

Strukturu této práce tvoří čtyři kapitoly. První kapitola je tento úvod. Druhá vymezuje základní pojmy potřebné k pochopení ostatních kapitol a vysvětluje obecný princip multimediálních frameworků a strukturu grafů filtrů. Rovněž popisuje některé významné frameworky. Součástí této kapitoly je také seznámení s frameworkem FFmpeg, které je nezbytné k pochopení této práce. Třetí kapitola se věnuje analýze požadavků na aplikaci, návrhu grafického uživatelského rozhraní, návrhu struktury. Popisu implementace a srovnání vytvořené aplikace s již existujícími nástroji se věnuje poslední kapitola. Vymezuje její výhody a nevýhody a uvádí možné směry pro pokračování vývoje.

Kapitola 1

Teoretický základ

Tato část bakalářské práce je zaměřena na vysvětlení základních pojmů k pochopení dalších částí. Obsahuje vymezení konceptu práce s grafy filtrů a srovnání několika multimediálních frameworků. Závěr kapitoly je zakončen shrnutím získaných poznatků a sumarizací existujících nástrojů pro tvorbu grafů filtrů.

1.1 Barevný model, formát pixelu, digitální video

Každý pixel obrazu má určitou barvu. Tuto barvu je možné jednoznačně definovat pomocí barevného modelu. Barevný model – je matematický model, který reprezentuje obsah základních barev ve výsledné barvě. Existuje několik barevných modelů. Nejvíce používaný je pravděpodobně model RGB. Tento model vychází z myšlenky, že lidské oko je citlivé na tři barvy červenou (R), zelenou (G) a modrou (B). Trojice těchto nezávislých složek určuje výslednou barvu. Dalším pozoruhodným modelem je např. YCbCr, který je transformací RGB modelu. S barevným modelem souvisí další, z hlediska této práce významný, pojem – formát pixelu. Formát pixelu – je použitý barevný model a způsob jeho uložení v paměti zařízení. [9][1]

Digitální video je diskrétní signál přenášející informaci o souřadnicích v prostoru a čase. Hodnotou jeho vzorku je barva. Kvůli extrémní paměťové náročnosti se s videem pracuje jako se sekvenci 2D snímků (angl. *frames*). [1]

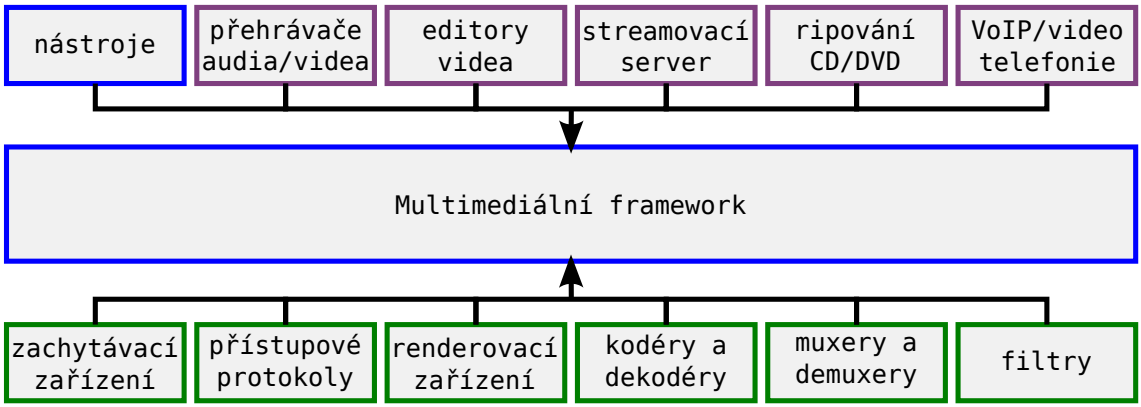
1.2 Multimédia, multimediální framework

V 21. století lze volně šířit informace. Různé informace lze různě ukládat, měnit a zpracovávat. Lze je také různě sdělovat. Právě se sdělováním informaci je spojen pojem *multimédia*.

Multimédia je slovo vzniklé spojením dvou anglických slov latinského původu – „*multi*“ (mnoho) a „*media*“ (sdělovací prostředek). V dnešní době lze multimédia definovat jako systém složený ze dvou částí. První část tvoří data, což mohou být zvukové signály, obrazy, videa, animace, textová data a mnoho dalších. Data jsou napojená na druhou část tohoto systému, na zařízení, které je schopno tato data reprezentovat, uchovat, zachytit. Tato zařízení jsou sdělovacími médii. Pokud je ovšem potřeba nejenom reprezentovat nebo ukládat data, musí existovat nástroj pro tyto účely. Takovým nástrojem může být rozhraní, které umožní práci s multimédii a ochrání uživatele před problémy s kompatibilitou a složitostí implementace. Tomuto rozhraní říkáme „framework“. Multimediální framework je soubor knihoven a nástrojů pro práci s multimediálními daty[1]. Framework v podstatě

tvoří nějaký základ pro různé aplikace a jejich požadavky na zpracování multimediálních dat. Framework může být platformně omezen, např. Video For Windows, nebo multiplatformní, např. FFmpeg. Většinou může být framework rozšířen o novou funkcionalitu. Pokud se jedná o open-source framework, může programátor upravit zdrojový kód podle svých potřeb. Není ovšem vždy nutno zasahovat do zdrojových kódů, drtivá většina multimediálních frameworků umožňuje vytvářet rozšiřující moduly speciálně navržené k určitým úkonům. Na obrázku 1.1 je znázorněna integrace zásuvných modulů do frameworku. Většinou se jedná o moduly zpřístupňující práci s novými formáty (kodéry a dekodéry) nebo zařízeními. Také to mohou být nové filtry obrazu, zvuku nebo nástroje pro zefektivnění práce s hardwarem, např. modul umožňující využít hardwarovou akceleraci při dekódování videa. Jak jsem již zmínil, existuje více způsobů přidání těchto modulů. Například framework GStreamer umožňuje načítání za běhu. Na druhou stranu vyžaduje framework FFmpeg znovupřeložení.

Mezi významné multimediální frameworky patří např. Video for Windows, QuickTime, DirectShow, Media Foundation, GStreamer a FFmpeg. Tyto frameworky jsou porovnány v dalších částech této kapitoly, a to po vysvětlení obecného konceptu uživatelského přístupu pro práci s frameworky. Jedná se o koncept „*grafů filtrů*“ [1].



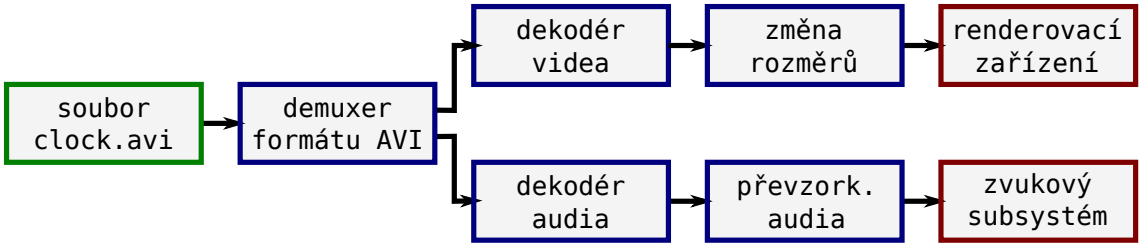
Obrázek 1.1: Integrace zásuvných modulů a přidání nové funkcionality do frameworku. Zdroj: [1]

1.3 Filtr, graf filtrů

Filtr je v pojetí multimédií modulem zaobalujícím nějakou funkcionalitu. Je nezbytnou součástí frameworku. Většinou lze filtry rozdělit do tří základních kategorií [1]:

- **Zdrojové** – jsou vstupním bodem toku dat, např. multimediální kontejner, do kterého je načten soubor uložený lokálně na disku nebo stažený z internetu. Zdrojovými filtry mohou být zachytávací/nahrávací zařízení.
- **Transformační** – provádí změny v těchto datech. Mezi transformační filtry se řadí (de)kodér, (de)muxer, filtry pro práci s obrazy, zvukovými signály, pro změnu formátu dat a mnoho dalších. Kodér dokáže převádět data z jednoho formátu do jiného (tzn. Zakódovat data). Muxer umožňuje sjednotit několik datových toků (obrazové a zvukové stopy, titulky) do jednoho datového toku. Demuxer naopak rozděluje jeden tok do více datových stop. Pro lepší pochopení těchto filtrů slouží obrázek 1.2. Transformační filtry také vytváří výstupní tok dat.

- **Výstupní** – poskytují rozhraní pro práci s výstupními toky dat (uložení do souboru, tisk, zobrazování na renderovacím zařízení).



Obrázek 1.2: Klasické zpracování vstupního souboru. Demuxer rozdělí vstupní datový tok na dva výstupní – audio a video tok. Příslušný dekodér se postará o čtení toku. Nad videem je aplikována změna rozměru a zvukový signál je převzorkován. Zdroj: [1]

Tyto filtry lze nalézt prakticky v každém multimediálním frameworku. Jak je patrné z obrázku 1.2, data tečou skrz filtry a samozřejmě se může stát, že se vlivem transformace změní jejich formát. V případě, že se formáty nebo typy dat na spoji dvou filtrů liší, filtry se jednoduše nespojí. Některé frameworky (např. FFmpeg) umožňují filtrům domluvit se na typu dat, umí tedy provádět omezené automatické typové konverze. Starší frameworky (Video for Windows) tuto vlastnost nemají. Pokud ale typy sedí, vzniká spojení a data proudí dále. Místa spojení dvou filtrů označujeme jako *pady*, nebo také *piny* v závislosti na frameworku. Pady mohou být vstupní nebo výstupní. Počet padů je definován typem filtru. Některé mají pouze jeden výstupní pad (typicky zdrojové filtry), jiné pouze jeden vstupní (výstupní filtry). Transformační filtry mají jeden a více vstupních a výstupních padů.

Některé frameworky mají fixní graf filtrů (Video for Windows), čímž se přesně definuje i způsob práce s frameworkem. Valná většina ovšem umožňuje vytvářet grafy filtrů přesně podle potřeby uživatele. V tom tkví síla frameworků jakožto nástroje pro programátory, kteří si tak mohou jednoznačně definovat, co všechno požadují od své aplikace. Může to být běžný přehrávač videa nebo streamovací server anebo konverter atd. Mnoho skvělých aplikací vzniklo právě díky tomu, že existoval vhodný multimediální framework. Příklady těchto aplikací jsou uvedeny u každého frameworku v následující části [1].

1.4 Významné multimediální frameworky

Tato část práce popisuje několik významných multimediálních frameworků a porovnává způsob tvorby grafů filtrů s obecným modelem definovaným v předchozí části. Prvním z probíraných frameworků je – Video for Windows. Tento multimediální framework není až tak podstatný z hlediska grafů filtrů. Mnohem podstatnějšími a plně využívajícími koncept grafů filtrů jsou frameworky DirectShow, MediaFoundation, GStreamer a samozřejmě FFmpeg. Důraz je kladen především na framework FFmpeg, poněvadž je z hlediska této práce nejdůležitější.

Video for Windows

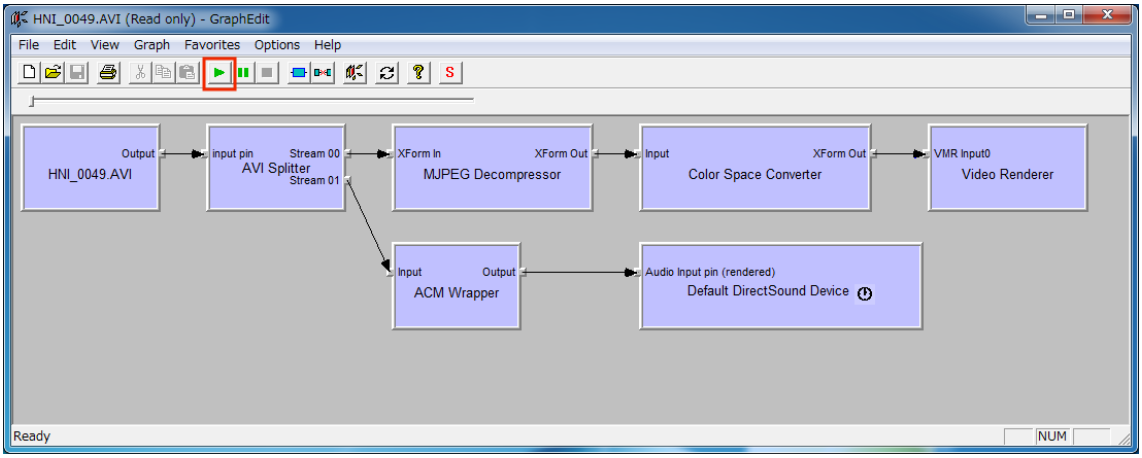
Video for Windows [6] byl vyvinut v roce 1992 společností Microsoft. Byla to určitá reakce na framework QuickTime od společnosti Apple. Bez rozšíření podporoval pouze soubory

typu AVI (Audio Video Interleave). Nad těmito soubory uměl provádět kompresi a dekompresi, přehrávání a zachytávání a několik jednoduchých operací. V dnešní době je tento framework označen za zastaralý, kvůli jednoduché implementaci pro něj však každý dnes používaný kodek implementuje modul [6].

DirectShow

DirectShow [4] je multimediální framework vyvinutý firmou Microsoft jako nástupce Video for Windows. Implementace je v jazyce C++ a je založena na COM (*Component Object Model*). Má značně rozšířenou funkcionalitu a také umožňuje vývojářům přistoupit k základním prvkům frameworku, čímž zjednodušuje rozšíření funkčnosti a tvorbu přídavných modulů. Další inovací oproti Video for Windows je automatická konverze barevných modelů.

Co se týče grafů filtrů, DirectShow umožňuje vytvářet prakticky libovolné grafy. Tyto grafy se skládají z filtrů tří základních typů (zdrojové, transformační a výstupní) a jsou vzájemně propojeny pomocí pinů. Je důležité, aby se propojené filtry dohodly na typu dat. V opačném případě spojení zanikne. Pro jednodušší tvorbu grafů filtrů obsahuje framework editační nástroj *GraphEdit* (viz. obrázek 1.3), který poskytuje intuitivní grafické rozhraní pro uživatele.

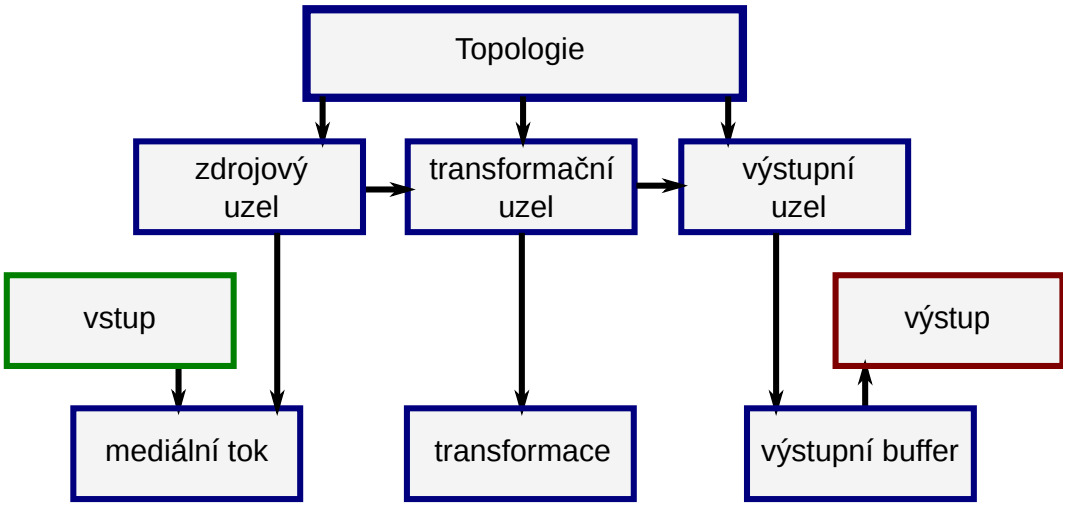


Obrázek 1.3: Grafické rozhrani nástroje GraphEdit.

V dnešní době je framework považován za zastaralý a byl nahrazen frameworkem Media Foundation [5].

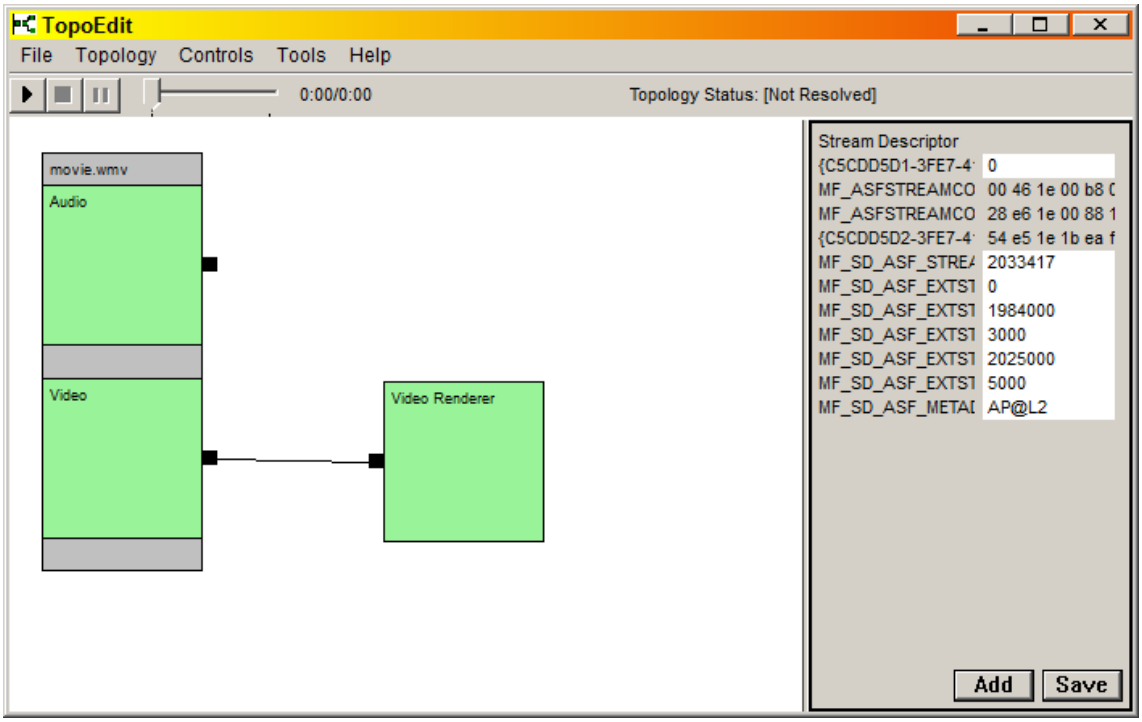
Media Foundation

Media Foundation [5] je multimediálním frameworkem, který nahradil zastaralý DirectShow. V základu je velice podobný DirectShow. Jádrem je znovu model COM a grafy filtrů. Graf filtrů je pojmenován topologie (*topology*) a filtru se říká uzel (*node*). Definovány jsou 4 typy uzlů – zdrojové, transformační, výstupní a nový typ filtrů *tee*. Tee uzly rozdělují datový tok více směry. Příklad této topologie je na obrázku 1.4.



Obrázek 1.4: Topologie v Media Foundation.

Uzly jsou vzájemně propojené. Pojmem *upstream* se označuje uzel poskytující data a *downstream* je uzel přijímající data. Stejně jako DirectShow, obsahuje framework editační nástroj – *TopoEdit* (viz. obrázek 1.5).



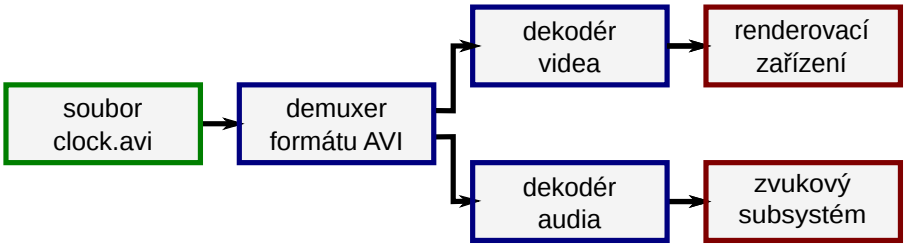
Obrázek 1.5: Grafické rozhraní nástroje TopoEdit.

Media Foundation je velice komplexní framework s širokou škálou podporovaných formátů a hardwarových zařízení a je stále aktuální (součást Windows SDK) [5].

GStreamer

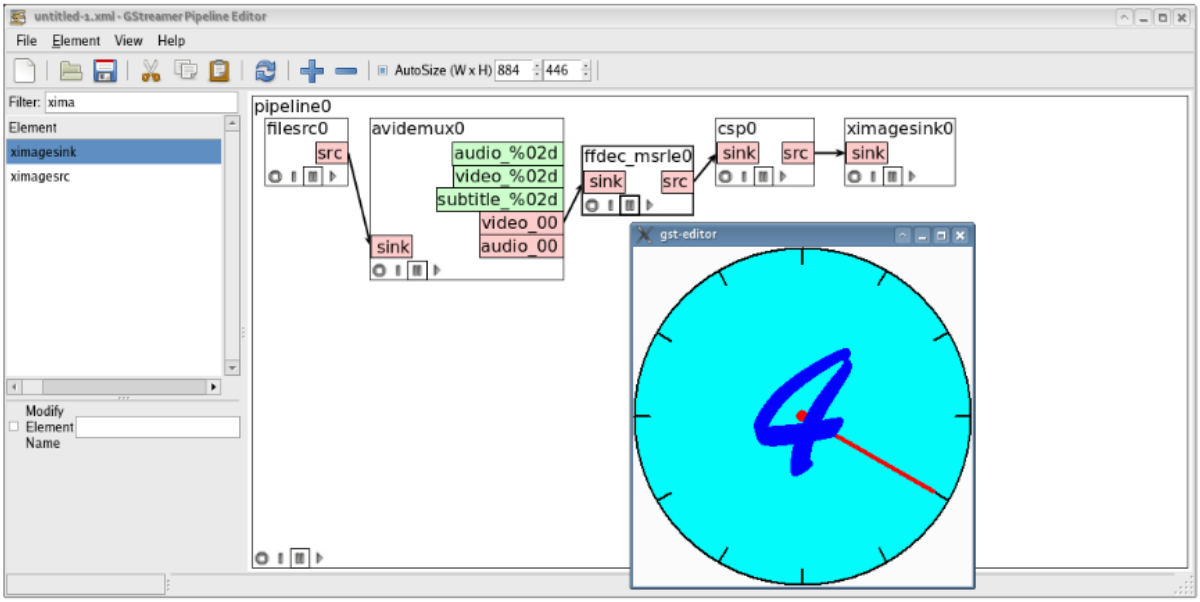
GStreamer [3] je open-source multiplatformní multimediální framework. Framework je založený na grafech filtrů označovaných pipeliny (*pipelines*). A jednotlivým filtrům se říká elementy (*elements*). Tyto elementy jsou propojeny pomocí propojných bodů, kterým se říká pady. Typy elementů odpovídají třem základním dříve definovaným typům.

Implementaci a načítání elementů obstarává příslušný zásuvný modul. Zásuvné moduly pro tento framework jsou dynamické knihovny. Jejich načítání probíhá za běhu. Framework je schopen propojovat elementy automaticky na základě dat protékajících těmito elementy. Pro lepší pochopení architektury GStreameru je na obrázku 1.6 příklad jednoduché pipeliny pro OGG přehrávač.



Obrázek 1.6: Pipelina pro jednoduchý přehrávač. Soubor je načten z disku. Pomocí demuxeru jsou rozděleny audio a video toky. Příslušné dekodéry dekódují toky a odešlou je na výstupní zařízení

Pro zjednodušení vytváření pipeline je k dispozici nástroj *gst-editor* (viz. obrázek 1.7).



Obrázek 1.7: Grafické rozhrani nástroje gst-editor. Zdroj: [1]

FFmpeg

Oficiální definice od tvůrců FFmpeg zní následovně: „*FFmpeg je vedoucí multimediální framework umožňující dekódování, kódování, transkódování, multiplexování, demultiplexování,*

streamování, filtrování a přehrávaní téměř všeho, co vytvořili lidé nebo stroje. Podporuje i nejvíce obskurní nejstarší formáty. Přičemž bez ohledu na to, zda byly vyvinuty standardovými výbory, komunitou nebo korporacemi. Rovněž je FFmpeg vysoce přenosný: FFmpeg může být sestaven, spuštěn a jeho testovací rozhraní FATE použito napříč systémy Linux, Mac OS X, Microsoft Windows, the BSDs, Solaris...“ [2]

Tato definice je velmi přesná, můžeme ji ovšem ještě doplnit. Jedná se o sadu knihoven pro zpracování audio a video datových toků. Používá LGPL licenci (některé části GPL). Rozhraní FFmpeg je implementováno v jazyce C. Tento framework využívá hodně významných projektů – MPlayer, VLC media player, Handbrake, atd. [2][1]

Z čeho se skládá FFmpeg

Jak bylo zmíněno výše, FFmpeg je tvořen několika knihovnami. Mezi nejvýznamnější patří následující [2]:

- **libavcodec** – obsahuje kodéry a dekodéry pro audio a video formáty.
- **libavutil** – je knihovnou zajišťující multiplatformní programování. Obsahuje funkce pro práci s textovými řetězci bezpečně přenosné na různé platformy, funkce pro generování náhodných čísel, konverzi barevných prostorů, práci s datovými strukturami, další matematické operace, kryptografii a převody číselných typů. Obsahuje rovněž definici důležitých formátů, datových proudů a enumeraci s formáty vzorků a pixelů.
- **libavformat** – poskytuje rozhraní pro multiplexování a demultiplexování audio a video datových toků a toků s titulky. Obsahuje muxery a demuxery pro různé multimediální kontejnery. Také implementuje několik vstupních a výstupních protokolů pro přístup k mediálním zdrojům.
- **libavfilter** – zajišťuje rozhraní pro vytváření a filtrování grafů filtrů. Obsahuje všechny základní filtry a struktury typu kontejner. V rámci této práce se jedná o nejvýznamnější knihovnu, na které je postaven systém tvorby grafů filtrů mezi uživatelem a FFmpeg frameworkem.
- **libavdevice** – je spojným bodem pro práci s množstvím multimediálních vstupních/-výstupních zařízení a frameworků (Video4Linux2, Video for Windows, DirectShow, ALSA, OSS, PulseAudio).
- **libswscale** – je knihovnou umožňující rychlou a kvalitní změnu formátů pixelů, velikostí obrázků a videa.
- **libswresample** – nabízí vysoce optimalizované funkce pro změnu formátu a vzorkovací frekvence audio datových toků.

Kromě knihoven jsou součástí balíků FFmpeg i konzolové nástroje [1][2]:

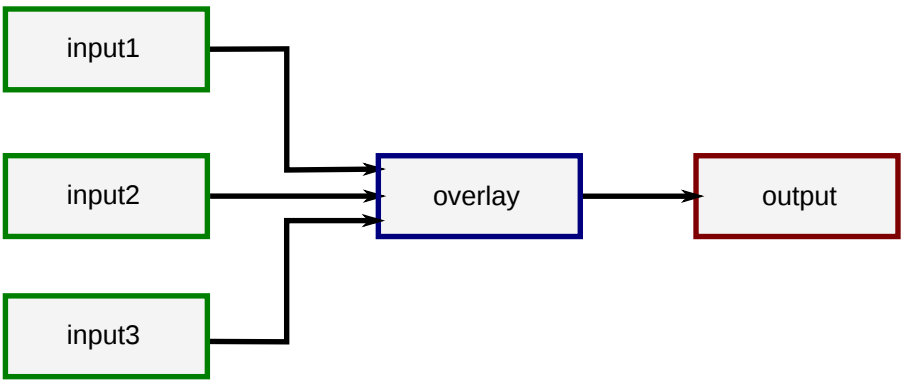
- **ffmpeg** – je velice rychlý audio a video konvertor. Podporuje získání vstupních dat z několika zdrojů. Mezi tyto zdroje patří klasické soubory, živá vysílání, roury, kamery atd. Počet vstupů ani výstupů není omezen. Cokoliv, co není argumentem příkazové řádky, je považováno za výstup. Tento nástroj také umožňuje filtrování.
- **ffprobe** – sbírá informace z různých datových toků a tiskne je v podobě vhodné pro člověka.

- **ffplay** – jednoduchý a přenosný přehrávač na bázi SDL.
- **ffserver** – streamovací server.

Grafy filtrů ve FFmpeg

FFmpeg nabízí velké množství filtrů jak pro video, tak i pro audio stopy. Tato práce se věnuje pouze video filtrům, ale použití audio filtrů je v podstatě stejné, samozřejmě krom názvů jednotlivých filtrů. Grafy filtrů FFmpeg se příliš neliší od obecného konceptu. Tyto grafy mohou být dvou typů [2]:

- **jednoduché** – mají jeden vstup a jeden výstup. Z hlediska této práce se zaměřuji pouze na jednoduché grafy filtrů. Proto budu níže popisovat právě jejich syntaxi. Příklad takového grafu je na obrázku 1.11.
- **komplexní** – mohou mít více vstupů a více výstupů. Příkladem takového grafu filtrů může být sloučení více vstupních toků a jejich překrytí do jednoho výstupního toku (viz. obrázek 1.8).



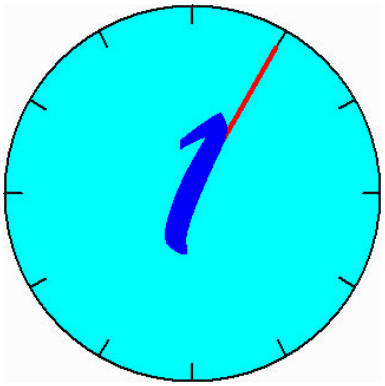
Obrázek 1.8: Komplexní graf.

Filtrům se říká *filtry* a místům jejich spojení *pady*. Konzolové nástroje používají textovou reprezentaci grafů filtrů. Proto je potřeba vysvětlit syntaxi těchto řetězců.

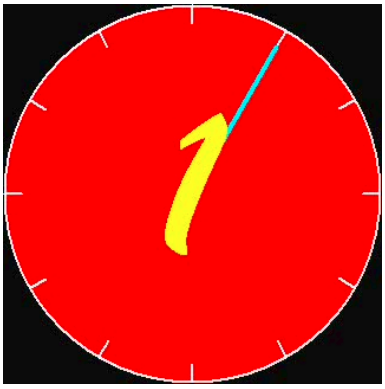
Nejdříve chci předvést použití jednoho filtru, například pomocí přehrávače *ffplay*. O rozpoznání, že se jedná o filtr, se stará argument *-vf*, za kterým následuje název filtru.

```
ffplay -vf negate clock.avi
```

V této ukázce je použit vzorový video soubor *clock.avi*. Na obrázku 1.9 je vyobrazen jeho první snímek. Na obrázku 1.10 je také první snímek tohoto videa, ovšem až po zpracování filtrem *negate*. Tento filtr zneguje hodnotu každé barevné složky v každém pixelu snímku. Výsledkem je, pochopitelně, změna každé barvy na inverzní barvu. Z bílé je černá, z červené – zelená apod.



Obrázek 1.9: Bez použití filtru negate.



Obrázek 1.10: S použitím filtru negate.

Když vznikne potřeba vytvořit složitější graf filtrů, musí se brát v potaz následující pravidla syntaxe:

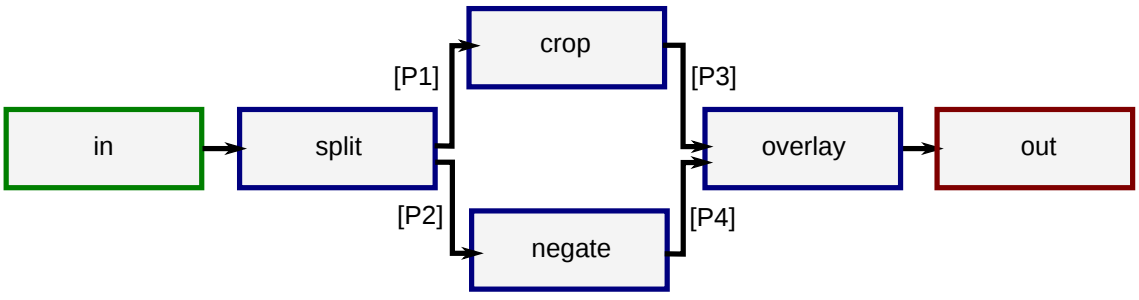
- Graf filtrů je tvořen jednotlivými řetězci filtrů.
- Tyto řetězce jsou odděleny středníkem („;“).
- Každý řetězec se skládá z jednotlivých filtrů oddělených čárkou („,“).

```
[vstup_1]...[vstup_N] filter =
argumenty[vystup_1]...[vystup_M]
```

- Filtr má následující syntaxi – *filter* je jméno filtru, *argumenty* je řetězec oddělených „:“ parametrů ve tvaru *klič=hodnota*, *vstup_1...vstup_N* jsou vstupní pady a *vystup_1...vystup_M* jsou výstupní pady. Pokud je nalezena dvojice stejně pojmenovaných vstupních a výstupních padů, jsou tyto pady propojeny. Pokud není k výstupnímu padu nalezen odpovídající vstupní pad, je tento výstupní pad navázán na první nepojmenovaný vstupní pad.

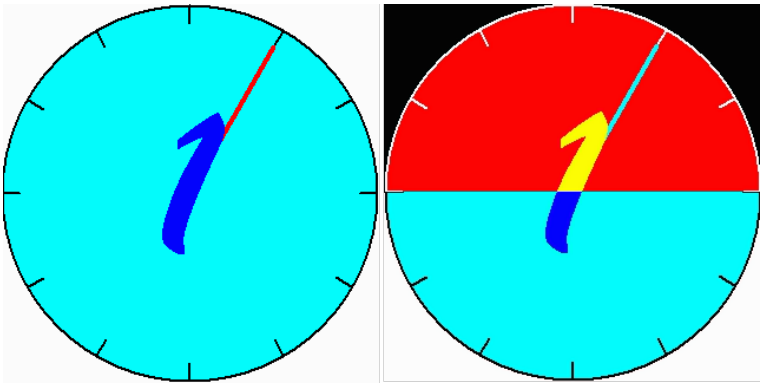
Pomocí těchto znalostí můžeme poskládat složitější graf filtrů. Jeho vizualice je na obrázku 1.11. V textové podobě bude vypadat následovně:

```
ffplay -vf "[in] split [P1][P2]; [P2] negate [P4]; [P1] crop=iw:ih/2:0:ih/2
[P3]; [P4][P3] overlay=0:H/2 [out]" clock.avi
```



Obrázek 1.11: Složitější graf FFmpeg.

Tento graf filtrů provede rozvětvení vstupního toku na dva toky. U prvního toku zneguje barvy. Druhý tok ořeže napůl a nechá pouze spodní polovinu. Následně sloučí tyto toky a překryje je. Ukázka aplikace tohoto grafu na testovacím videu je na obrázku 1.12.



Obrázek 1.12: Klíčový snímek s a bez použití složitějšího filtru.

Pro vizualizaci grafů filtrů existuje konzolový nástroj *graph2dot*. Tento nástroj převádí řetězec s grafem filtrů na jeho obrázkovou reprezentaci. Ukázka grafu filtrů z obrázku 1.11 byla vygenerována právě tímto nástrojem [1].

1.5 Shrnutí multimediálních frameworků

V popisovaných frameworcích lze s jistotou určit souvislosti v použití konceptu grafů filtrů. Model, který je popsán jako obecný případ, v podstatě definuje základ pro všechny způsoby práce s grafy filtrů. Někde se grafům filtrů říká pipeline, někde topologie. Filtry se označují za elementy, uzly nebo jednoduše filtry a jsou propojené pomocí padů nebo pinů. Rozdíly v těchto modelech nejsou až tak významné.

Většina frameworků nabízí uživateli nástroj s grafickým rozhraním pro sestrojení grafů filtrů. DirectShow má nástroj *GraphEdit* (obr. 1.3), GStreamer – *gst-editor* (obr. 1.7) a MediaFoundation má *TopoEdit* (obr. 1.5). Všechny tyto nástroje jsou velmi intuitivní a umožňují po sestrojení grafu filtrů přehrát náhled. Výjimkou je FFmpeg s nástrojem *graph2dot*. Ten totiž nabízí zpětnou možnost – vizualizovat již sestrojený graf, což je opačnou a spíše pouze informativní možností. Software s grafickým uživatelským rozhraním pro tvorbu grafů filtrů pro FFmpeg vážně chybí na trhu, a právě z tohoto důvodu se mu věnuje tato práce.

Kapitola 2

Analýza požadavků a návrh řešení

Tato kapitola se věnuje shrnutí požadavků na funkcionalitu aplikace, návrhu vzhledu frontendu a návrhu implementace backendu. Kromě toho, po analýze požadavků, popisuje vybraný grafický framework a jeho klíčové vlastnosti. Co se týče inspirace již existujícími řešeními, posloužily tomu nástroje zmiňované v předchozí kapitole (GraphEdit, gst-editor).

2.1 Analýza požadavků

Na aplikaci jsou kladeny následující požadavky:

- umožnit načíst a uložit video soubor ve formátu podporovaném FFmpeg
- vytvářet a mazat filtry, nastavovat jejich parametry
- vkládat filtry na pracovní plochu pomocí *Drag and Drop* (přetažení myši)
- spojovat filtry, mazat spojení
- zobrazovat náhled zpracovaného videa
- načítání a exportování grafů filtrů ve formátu textového řetězce pro ffplay

Naopak předem definovaným omezením je podpora více vstupů a výstupů.

2.2 Rozbor struktury FFmpeg

V této sekci je vysvětlena část vnitřní stavby frameworku FFmpeg podstatná k pochopení bakalářské práce. Popsány jsou především struktury a funkce spojené s filtry a grafy filtrů. Při výběru a popisu jednotlivých funkcí a struktur některé informace, jsou části a prvky zanedbány, jelikož struktura FFmpeg je velice obsáhlá.

Načítání, (de)kódování, ukládání

Před prací s FFmpeg je nutno zavolat funkci `av_register_all`, která zaregistruje všechny potřebné kodeky. Jakýkoliv graf filtrů začíná vstupním tokem dat. V rámci této práce je podstatné načítaní souborů z disku. K tomu slouží funkce `avformat_open_input`, která přečte soubor a zaobalí potřebné informace do struktury `AVFormatContext`, která obsahuje 2.1:

```
AVFormatContext{
...
    char filename [1024]      // jméno souboru
    AVInputFormat * iformat   // vstupní kontejnerový formát
    AVOutputFormat * oformat  // výstupní kontejnerový formát
    AVStream ** streams       // seznam datových toků
    int  bit_rate             // datový tok/přenosová rychlost
    int64_t duration          // délka trvání
...
}
```

Zdrojový kód 2.1: Struktura AVFormatContext.

Tento kontext slouží i k zápisu do souboru. Pro jeho korektní uvolnění slouží funkce `avformat_close_input`. Seznam toků `streams` obsahuje jednotlivé datové stopy a uvnitř každé z nich je struktura typu `AVCodecContext` 2.2. V ní jsou uložené klíčové informace o kodeku.

```
AVCodecContext{
...
    AVMediaType  codec_type    // typ kodeku
    AVRational   time_base     // jednotka času pro časová razítka
    int width                  // výška obrázku
    int height                 // šířka obrázku
    AVPixelFormat pix_fmt      // formát pixelů
    int sample_rate            // vzorkovací frekvence
...
}
```

Zdrojový kód 2.2: Struktura AVCodecContext.

Nejdůležitější z nich je `codec_type` určující, zda se jedná o video nebo audio stopu. Pomocí funkce `avcodec_find_decoder` a struktury `AVCodecContext` lze najít samotný dekodér. A funkce `avcodec_find_encoder` vrátí kodér. Inicializace probíhá voláním funkce `avcodec_open2`. Pro čtení existuje funkce `av_read_frame`, která načítá jeden paket `AVPacket`. Tento paket je pomocí funkce `avcodec_decode_video2` dekóduje na snímek `AVFrame`, který může být upraven a následně zakódován zpátky pomocí `avcodec_encode_video2`.

Pří ukládání je nejdříve pomocí funkce `avformat_alloc_output_context2` alokován výstupní `AVFormatContext`. Funkce `avformat_write_header` zapíše potřebné informace o stopě a každý zakódovaný snímek se zapíše pomocí `av_write_frame`. Zápis do souboru musí být ukončen funkcí `av_write_trailer` [2][1].

Filtry

Pro práci s filtry je důležitá funkce `avfilter_register_all`, která musí být volána na začátku programu, aby bylo možné používat všechny zabudované filtry. Samotný filtr je struktura typu `AVFilter`. V rámci této práce je důležitá tato její část 2.3:

```
AVFilter{
...
  const char * name
  const char * description
  AVFilterPad * inputs
  AVFilterPad * outputs
  AVClass * priv_class
  AVFilter * next
...
}
```

Zdrojový kód 2.3: Struktura AVFilter.

Kde `name` je jméno filtru, `description` je popis filtru a `inputs/outputs` jsou seznamy vstupních a výstupních padů (vzhledem k vybrané metodě sestrojení grafů filtrů nejsou až tak podstatné). Ukazatel na následující filtr `next` umožňuje projít všechny zabudované filtry pomocí funkce `avfilter_next` a zjistit důležité vlastnosti filtrů, které jsou ukryté ve struktuře typu `AVClass`. Struktura `AVClass` vypadá následovně 2.4:

```
AVClass{
...
  const char * class_name
  AVOption * option
  AVClass * child_class_next
...
}
```

Zdrojový kód 2.4: Struktura AVClass.

Položka `class_name` je jméno třídy, `option` je ukazatel na první strukturu `AVOption`, která v případě sounáležitosti s filtry představuje první argument filtru. Tato struktura uchovává jméno argumentu, jeho typ, rozsah hodnot a výchozí hodnotu. Struktura obsahuje ukazatel `child_class_next`, který udává další třídu patřící stejnému filtru [2][1].

Grafy filtrů

Grafy filtrů v FFmpeg lze vytvářet ručně, vytvořením jednotlivých filtrů, nastavením potřebných příznaků a parametrů, propojením jednotlivých padů a následným přidáním do grafů filtrů. V rámci této práce, jejímž účelem je vytvořit grafické uživatelské rozhraní, jsem se rozhodl pro použití funkce `avfilter_graph_parse`. Tato funkce umí vytvářet grafy filtrů z řetězce filtrů ve formátu nástroje *ffmpeg*. Tento graf obdrží pomocí funkce `av_buffersrc_add_frame_flags` snímek `AVFrame`, zpracuje jej a následně vrátí pomocí funkce `av_buffersink_get_frame` [2][1].

2.3 Výběr a popis GUI frameworku

Vzhledem k tomu, že FFmpeg je multiplatformní multimediální framework a je implementovaný v jazyce C, bylo rozhodnuto použít *Qt framework*[1] [7]. Qt umožňuje programování

[1] Dostupné z https://download.qt.io/

v jazyce C++ a je také multiplatformní. Toto rozhodnutí bylo ovšem ovlivněno osobními zkušenostmi s tímto frameworkem.

Qt nabízí celou škálu grafických objektů označovaných *widgety*. Tyto widgety jsou základem pro tvorbu grafického uživatelského rozhraní. Mezi nejdůležitější z těch, které hodlám použít, patří [8]:

- **QMainWindow** – reprezentuje hlavní okno aplikace.
- **QDockWidget** – kontejner. Může být přemístěn do jedné z předem povolených pozic v hlavním okně. Uživatel může měnit velikost tohoto widgetu, skrýt jej nebo znovu zobrazit.
- **QPushButton** – jednoduché tlačítko.
- **QLabel** – ikona, může obsahovat text, obrázek nebo barevnou výplň.
- **QGraphicsScene** – plátno umožňující přidávat na něj jiné objekty.
- **QGraphicsView** – pohled na QGraphicsScene. Určuje velikost zobrazené části plátna, umožňuje zoomování, pohyb v rámci plátna.
- **QGraphicsItem** – grafický objekt (obrázek, text, geometrický tvar).
- **QMenu** – jednoduché menu s možností větvení položek a zanořování.
- **QToolBar** – lišta s prvky vyvolávajícími rychlé akce.

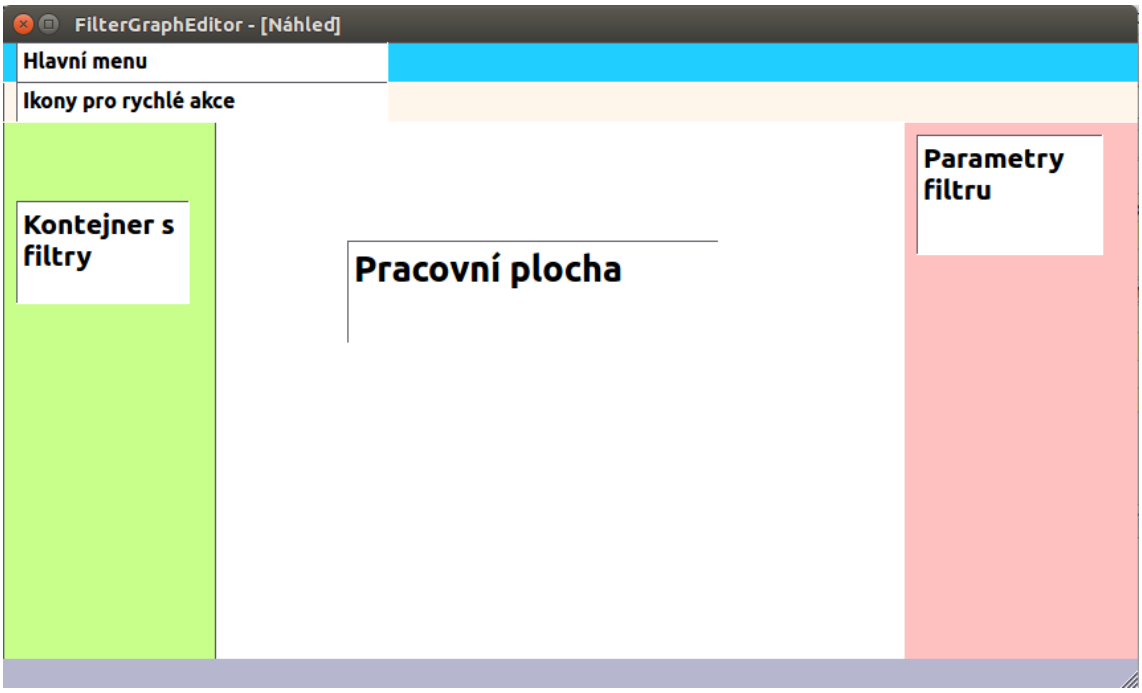
V rámci implementace jsou využité i další Qt objekty. Ovšem pro účely pochopení této práce nejsou příliš podstatné.

2.4 Návrh frontendu

Při navrhování grafického uživatelského rozhraní je nutno si nejdříve uvědomit, které prvky jsou v něm nezbytné, a které naopak přináší narušení jednoduchosti a intuitivního uživatelského ovládaní. Podle mého existuje pět základních částí, které GUI musí obsahovat. Mezi ně patří:

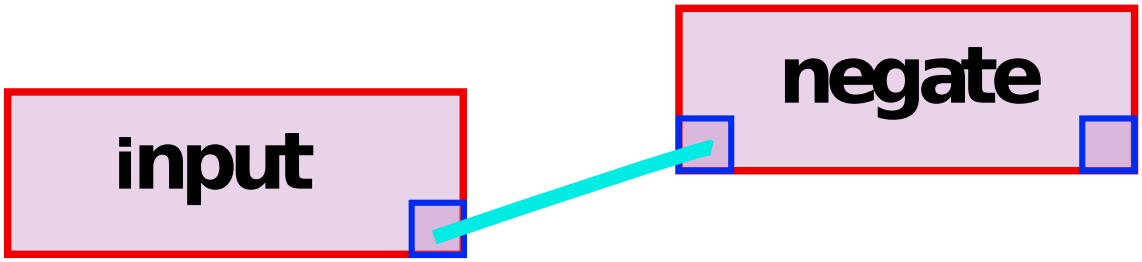
- **hlavní menu** – reprezentací je pruh s rozbalovacími položkami. Mezi základní patři – „Soubor“ s vnořenými položkami pro manipulaci s vstupy/výstupy, “Úpravy” k provedení některých operací nad grafem filtrů, „Zobrazení“ umožňující skrýt jednotlivé části, případně „Nápověda“.
- **ikony rychlých akci** – obrázkové ikony provádějící některé často používané operace.
- **seznám filtrů** – kontejner obsahující seznam všech podporovaných filtrů.
- **parametry filtrů** – kontejner s tabulkou pro nastavení parametrů aktuálně vybraného filtru.
- **pracovní plocha** – plátno umožňující přidávat na něj filtry ze seznamu filtrů. Přidání musí být uživatelsky intuitivní. Proto hodlám implementovat přidání pomocí přetažení myši (Drag and Drop) a také pomocí dvojkliku. Přidané filtry musíme být také schopni vybrat, odstranit, přemístit kamkoliv v rámci plátna.

Rozložení těchto prvků je znázorněno na obrázku 2.1.



Obrázek 2.1: Návrh frontendu.

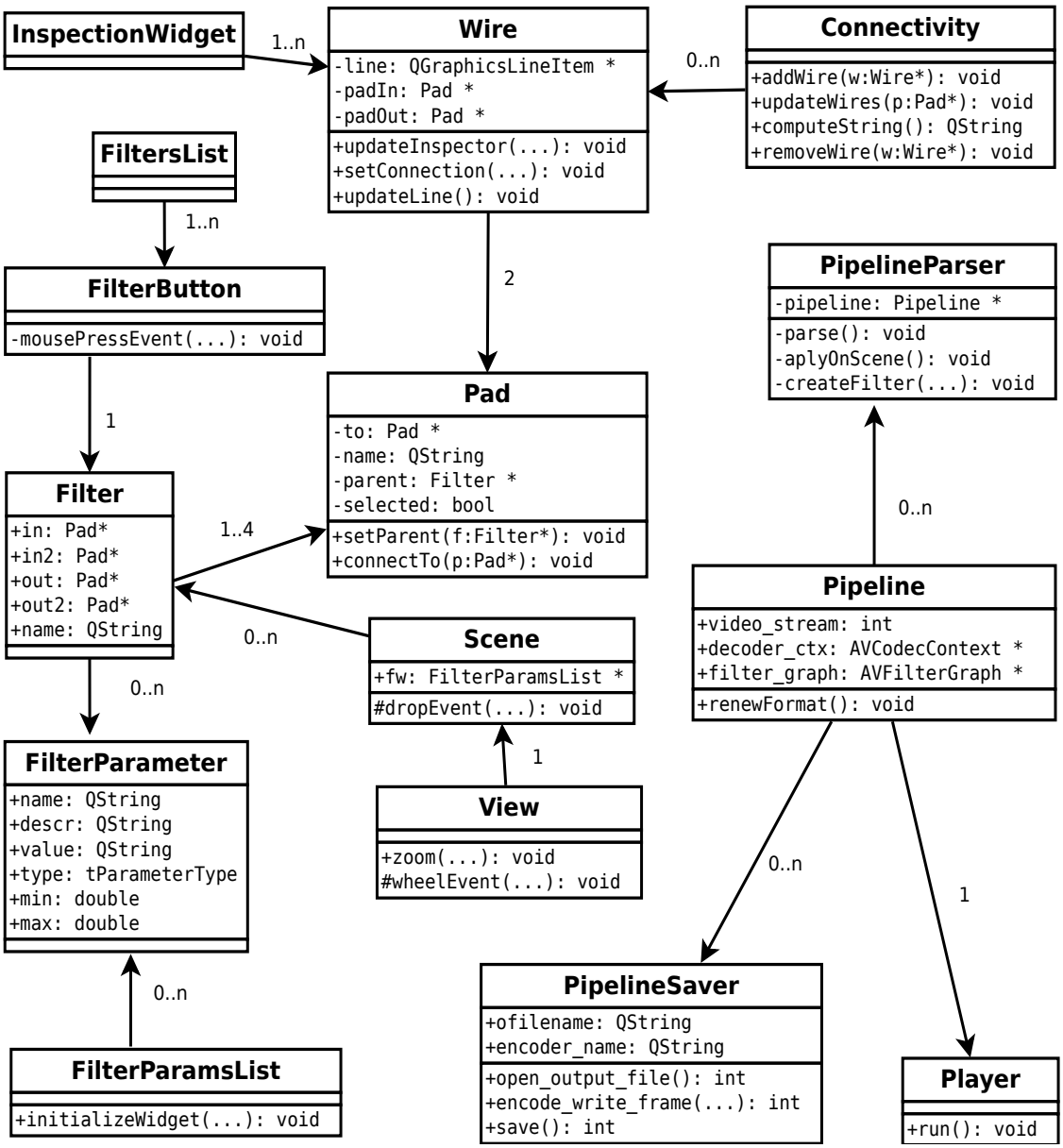
Jednotlivé filtry si představuji jako obdélníky se jmény filtrů a čtverce znázorňující pady. Jejich propojení bude znázorněno čárou. Pro lepší představu slouží obrázek 2.2.



Obrázek 2.2: Návrh filtrů a jejich propojení.

2.5 Návrh backendu

Tato část se zabývá návrhem struktury aplikace, vysvětlením základních tříd a jejich vztahu k modelu grafů filtrů, který byl uveden v teoretické části. Nejlepším způsobem, jak popsat návrh backendů, je vysvětlit zkrácený diagram tříd z obrázku 2.3.



Obrázek 2.3: Zjednodušený diagram tříd.

Význam jednotlivých tříd a namapování na obecný model práce s grafy filtrů je popsán v tabulce 2.4.

Třída	V obecném modelu	Význam
Scene	–	Třída reprezentující pracovní plochu. Na ní jsou pomocí myši přetaženy jednotlivé filtry.
View	–	Umožňuje přibližování a oddalování (zoomování). Posun po pracovní ploše.
Connectivity	–	Zajišťuje zpracování, vytvoření a mazání spojení mezi filtry, sestavuje textový řetězec grafu filtrů.
Pad	pad/pin	Místo spojení dvou filtrů. Může být buď vstupní nebo výstupní.
Filter	filtr/element	Základní kámen pro tvorbu grafů filtrů, obsahuje seznam vstupních a výstupních padů, umístění na pracovní ploše a typ.
Wire	spojení dvou filtrů	Uchovává v sobě informace o jednom spojení.
FilterParameter	argument filtru	Zaobaluje argument filtru.
FilterParamsList	–	Tabulka umožňující editaci hodnot parametrů filtru.
FilterButton	–	Znázorňuje filtr. Pří přesunutí na pracovní plochu spouští Drag&Drop obsluhu.
FiltersList	–	Kontejner reprezentující seznam dostupných filtrů.
InspectionWidget	–	Tabulka s výsledky dynamické inspekce.
Pipeline	pipeline/graf filtrů	Zaobaluje vstupní kontejner a vytvořený graf filtrů.
PipelineParser	–	Provádí sestavení grafů filtrů z textové podoby.
PipelineSaver	–	Ukládá výstup pipeliny do souboru.
Player	–	Přehrávač náhledu.

Tabulka 2.4: Význam jednotlivých tříd a namapování na obecný model.

Kapitola 3

Implementace a dosažené výsledky

Tato kapitola se zabývá popisem implementované funkcionality a významem jednotlivých tříd v rámci projektu. Také vysvětluje propojení mezi vlastním rozhraním a použitím knihoven FFmpeg. Kapitola je ukončena shrnutím dosažených výsledků (popisem vytvořené aplikace) a porovnáním s jinými existujícími řešeními.

3.1 Zaobalení FFmpeg struktur

V předchozí kapitole byla popsána návaznost na obecný model grafů filtrů. Ovšem integrace struktur FFmpeg zatím popsána nebyla. Tabulka 3.1 vysvětluje způsob použití FFmpeg struktur v rámci projektu. Hodnoty ve sloupci „*Způsob integrace*“ uvádí jeden ze tří použitých přístupů začlenění těchto struktur. V případě, že třída pouze významově odpovídá struktuře, je použitým způsobem „*znázorňuje*“. Způsob „*zaobaluje*“ znamená, že odkaz na strukturu je jednou z proměnných instancí třídy. V případě, že část funkcionality nebo vlastnosti byla převzata, je použit způsob integrace „*přejímá*“.

Třída	Způsob integrace	FFmpeg struktura
Pad	znázorňuje	AVFilterPad
Filter	přejímá	AVFilter
Filter	zaobaluje	AVFilterContext
Wire	znázorňuje	AVFilterLink
FilterParameter	přejímá	AVOption
Pipeline	zaobaluje	AVFormatContext, AVFilterGraph, AVCodecContext

Tabulka 3.1: Začlenění FFmpeg struktur do vlastní implementace.

3.2 Popis jednotlivých tříd

V následujících částech této sekce jsou popsány jednotlivé třídy, jejich význam a klíčové vlastnosti. Důležité metody jsou doplněné ukázkou zdrojového kódu.

Třída MainWindow

Předkem třídy *MainWindow* je třída QMainWindow. Instance této třídy je hlavní uživatelské okno obsahující v horní části menu QFileMenu a lištu QToolBar s ikonami pro rychlé akce QAction. Lišta obsahuje ikony pro následující funkcionalitu:

- **načtení grafu filtrů**
- **uložení grafu filtrů**
- **uložení zpracovaného videa**
- **přehrání náhledu**
- **zobrazení textové reprezentace grafu filtrů**
- **automatické propojení grafu filtrů**
- **propojení dvou padů**
- **smazání označeného prvku**
- **smazání všech prvků na pracovní ploše**
- **inspekci vlastností označeného prvku**
- **změnu způsobu ovládání pracovní plochy**
- **vyvolání nápovědy k označenému prvku**

Příklad vytvoření akce QAction demonstruje úsek zdrojového kódu 3.1.

```
...
    clear = new QAction(tr("&Clear␣canvas."), this);
    clear->setIcon(QIcon("clear.png"));
    clear->setShortcut(Qt::Key_C);
    clear->setStatusTip(tr("Clear␣canvas."));
    connect(clear, SIGNAL(triggered()), this, SLOT(clearAll()));
...
```

Zdrojový kód 3.1: Implementace vytvoření rychlé akce QAction a navázaní signálu vyvolání akce na odpovídající slot.

Všechny rychlé akce jsou pomocí mechanismu signálů a slotů napojené na sloty vykonávající příslušné operace. To znamená, že třída MainWindow zpracovává uživatelské vstupy a zahajuje příslušnou funkcionalitu v backendu aplikace.

Třídy Scene, View

Třída *Scene* je potomkem třídy QGraphicsScene. Její instance slouží jako pracovní plocha, na kterou lze umisťovat jiné grafické objekty. Dodatečně byla implementována obsluha Drag&Drop události. Implementaci této obsluhy popisuje zdrojový kód 3.2. Třída *View* dědí funkcionalitu třídy QGraphicsView a znázorňuje pohled na pracovní plochu. Navíc tato třída zpřístupňuje posun v rámci pracovní plochy, přiblížení a oddalování, které bylo vylepšeno o plynulou animaci a navázáno na událost pohybu kolečka myši.

```
void Scene::dropEvent(QGraphicsSceneDragDropEvent *event)
{
    QString filterName(event->mimeData()->text());

    if ( filterName.compare("input") == 0 ||
         filterName.compare("output") == 0 )
    {
        for(auto f : filters)
            if(f->getName().compare(filterName) == 0) return;
    }

    QPointF pos(event->scenePos()-QPointF(100,25));
    createNewFilter(filterName, pos);
}
```

Zdrojový kód 3.2: Implementace obsluhy drop události třídy Scene.

Třídy Pad, Filter, Wire

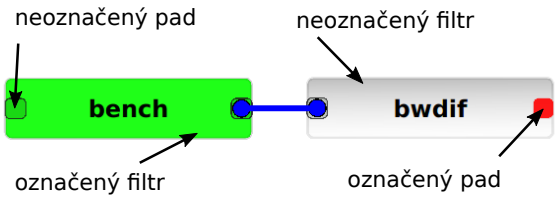
Třídy *Filter* a *Pad* jsou potomky třídy QGraphicsItem. Třída *Wire* v sobě udržuje objekt QGraphicsItem. Proto jsou instance těchto tříd zastoupeny na pracovní ploše (instancí třídy Scene).

Třída Pad v sobě uchovává unikátní identifikátor `QString name`, který je padu přiřazen při vytvoření. Dále obsahuje odkaz na rodičovský objekt `Filter * parent` (instanci třídy Filter) a odkaz na propojený pad `Pad * to`, který v případě nepropojeného padu obsahuje hodnotu `NULL`. Posledním důležitým parametrem třídy je pravdivostní hodnota `bool selected` uchovávající informaci o tom, zda byl pad označen uživatelem na pracovní ploše. Funkce `void paint(...)` zděděná od třídy QGraphicsItem a přepsaná pro vlastní účely zajišťuje vykreslování. V případě, že pad nebyl označen uživatelem, vykresluje pad jako šedý čtverec se zaoblenými okraji. V opačném případě je barva čtverce červená. Vykreslení padu je znázorněno na obrázku 3.2. Instance třídy pad jsou vytvářeny spolu s rodičovským objektem typu Filter a zanikají spolu s ním. Kromě konstruktoru a destruktoru obsahuje třída Pad i jiné funkce, nejdůležitější z nich jsou následující:

- `void setParent(Filter * f)` – nastaví rodičovský objekt
- `Filter * getParent()` – vrací rodičovský objekt
- `void connectTo(Pad * p)` – propojí pady
- `void setConnectivity(Connectivity * c)` – nastaví třídu konektivity

- `QString getName() const` – vrací unikátní jméno padu
- `void setName(const QString& name)` – nastaví unikátní jméno padů

Třída Filter znázorňuje FFmpeg strukturu AVFilter zaobalenou do grafického prvku. Vytvoření instancí této třídy probíhá ve funkci obsluhující událost mechanismu Drag&Drop ve třídě Scene. Funkce zajišťující vykreslování `void paint(...)` byla opět přepsána pro vlastní účely. Způsob vykreslování je na obrázku 3.2.



Obrázek 3.2: Označení filtrů a jejich padů.

Mezi nejdůležitější parametry a funkce třídy patří:

- `QString name` – jméno filtru, odpovídá jménu filtru ve FFmpeg
- `Pad * in,in2,out,out2` – odkazy na vstupní a výstupní pady
- `QString description` – krátký popis vlastností filtru
- `void setCtx(AVFilterContext * context)` – nastaví filtru jeho filtrovací kontext získaný z grafu filtrů FFmpeg (struktura AVFilterGraph)

Třída Wire představuje propojení AVFilterLink. Obsahuje odkazy na propojené pady – `Pad * padIn` a `Pad * padOut`. Dále obsahuje funkci `setConnection(...)`, která propojí pady. Dalším z jejích parametrů je grafický prvek, který tuto třídu zastupuje – `QGraphicsLineItem * line`. Funkce `updateLine()` překreslí tento prvek (úsečku) v případě, že došlo ke změně polohy jednoho z propojených padů. Poslední je funkce `updateInspector(QTableWidget * w)`, která nastaví dynamická data v případě inspekce.

Třída Connectivity

Jak název napovídá (z angl. *connectivity* = propojení), třída *Connectivity* zajišťuje správu propojení. Obsahuje seznam aktuálních propojení – `QList<Wire*> wires`, funkce pro přidávání spojení `void addWire(...)` a rušení spojení `void removeWire(...)`. Navíc pomocí aktuálních propojení umí sestavit textovou podobu grafů filtrů pomocí funkce `QString computeString()`.

Třída FilterParameter

FFmpeg struktura AVOption v sobě uchovává informace o argumentu filtru. V rámci této práce ji reprezentuje třída *FilterParameter*. Tato třída má šest parametrů:

- `double min` – dolní hranice hodnoty argumentu
- `double max` – horní hranice hodnoty argumentu
- `QString descr` – krátký popis argumentu
- `QString name` – jméno argumentu

- `QString value` – aktuálně nastavená hodnota
- `ParameterType type` – typ hodnoty (pravdivostní hodnota, celočíselná, textový řetězec atd.)

Třídy Pipeline, PipelineParser, PipelineSaver

Třída *PipeLine* zaobaluje vstupní kontejner AVFormatContext a jeho dekodér AVCodecContext. Také obsahuje graf filtrů AVFilterGraph. Třídy *PipelineParser*, *PipelineSaver* pracují s aktuální instancí třídy Pipeline.

Instance třídy PipelineSaver umožňuje uživateli uložit graf filtrů do textového souboru pomocí funkce `saveToTxt()` a do souboru formátu XML pomoci funkce `saveToXml()`. V případě uložení do formátu XML jsou filtry serializovány s parametry typu – jméno, pozice na pracovní ploše, jména padů, seznam parametrů filtru ve tvaru klíč-hodnota. Dále jsou ukládána jednotlivá propojení s informací o jménech propojených padů. Další možností je ukládání upraveného videa do souboru pomocí funkce `save()`.

Třída PipelineParser umožňuje sestavit graf filtrů z textového řetězce, pomocí funkce `parse()` a FFmpeg funkce `avfilter_graph_parse(...)`. Po sestavení tohoto grafu použije funkce `applyOnScene()` jeho strukturu a vytvoří podle ní graf filtrů na pracovní ploše. Taktéž se instance třídy PipelineParser stará o sestavení grafů filtrů pomoci souboru XML. Funkce `parseXml()` nejdříve vytváří jednotlivé filtry umístěné do správných pozic a následně je propojuje.

Třídy FilterList, FilterButton

Kontejner QDockWidget je předkem třídy *FilterList*. Do ní jsou vkládány instance třídy *FilterButton*, které jsou potomky Qt widgetu QPushButton. Seznam přístupných filtrů je načten pomocí funkce `avfilter_next(AVFilter*)`, která v cyklické smyčce postupně vrací jeden z dostupných filtrů AVFiter. Podle počtu padů a názvu se pomocí funkce `createPixmap()` vytvoří ikona QPixmap, která znázorňuje příslušný filtr. Rovněž třída FilterButton obsahuje obsluhu události `mousePressEvent()`, která vyvolá přesun typu QDrag. Kód 3.3 ukazuje implementaci této obsluhy.

```
void FilterButton::mousePressEvent(QMouseEvent *e){
    QDrag *drag = new QDrag(this);
    QMimeData *mime = new QMimeData;
    drag->setMimeData(mime);
    mime->setText(this->property("name").toString());
    drag->setPixmap(createPixmap());
    drag->setHotSpot(QPoint(100, 25));

    drag->exec();
}
```

Zdrojový kód 3.3: Implementace obsluhy události zmáčknutí tlačítka FilterButton a započetí události QDrag.

Třídy FilterParamsList, InspectionWidget

Třídy *FilterParamsList* a *InspectionWidget* dědí funkcionalitu třídy QDockWidget a obsahují tabulku QTableWidget.

Třída FilterParamsList obsahuje seznam parametrů filtru a umožňuje jejich editaci. Při výběru filtru na pracovní ploše je tabulka inicializována podle seznamu parametrů vybraného filtru. Proběhne nastavení ošetření uživatelských vstupů pomocí instance třídy QRegExpValidator a při jakékoliv změně buňky tabulky je aktuální hodnota uložena v příslušném parametru FilterParameter.

Instance třídy InspectionWidget zjišťuje data o spojení filtrů během přehrávání náhledu. Podrobněji bude tento mechanismus popsán při vysvětlení implementace přehrávače.

Třída Player

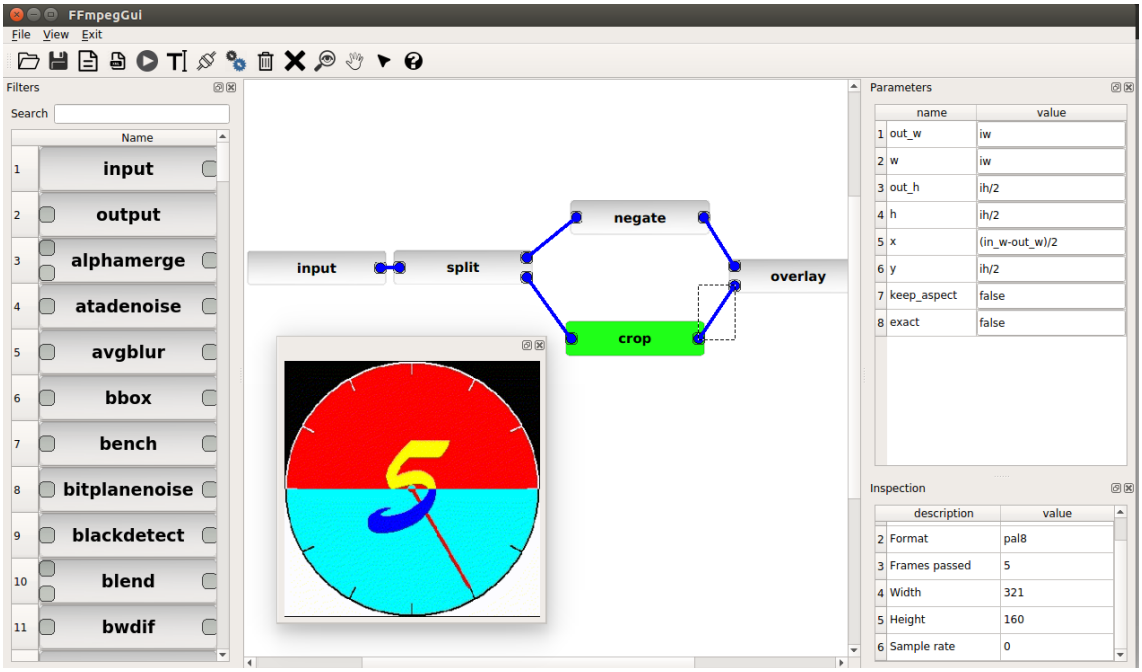
Třída *Player* je potomkem Qt třídy *QObject*. Tato dědičnost umožňuje přesunout funkcionalitu její instance do jiného vlákna (instanci třídy *QThread*). Znázorňuje přehrávač náhledu. Obsahuje následující důležité parametry, funkce, signály a sloty:

- `QPushButton* btn` – tlačítko sloužící jako zobrazovací plocha
- `bool stoped` – pravdivostní hodnota s informací, zda bylo přehrávání pozastaveno
- `void stopedSlot()` – slot funkce, která se vykoná v případě pozastavení přehrávání
- `void frameFinishedSig()` – signál, že jeden snímek byl úspěšně přečten
- `void procDone()` – signál oznamující, že lze pokračovat ve čtení
- `void playFinished()` – signál oznamující, že čtení snímku bylo dokončeno
- `void run()` – hlavní smyčka, ve které probíhá čtení, dekódování a zobrazení snímků.

Při vytváření přijímá odkaz na Pipeline objekt, ze kterého zjišťuje aktuální graf filtrů a vstupní kontejner. Následně pomocí funkce `av_read_frame()` čte video snímky z kontejneru. Každý snímek je dekódován a odeslán grafu filtrů. Jakmile se snímek objeví na výstupu grafu filtrů, je vyslán signál `void frameFinishedSig()` oznamující instanci třídy InspectionWidget v hlavním vlákně, že si může aktualizovat svoje data. Přehrávání je pozastaveno. Po aktualizací dat je vyslán signál, který přehrávač příjme a vyšle svůj interní signál `void procDone()`, který znovu spustí přehrávání. Graf filtrů vrací snímek ve formátu YUV420. Tento formát je převeden na RGB24 a uložen do obrázku typu QImage. Pomocí snímkové frekvence je vypočítána doba po jejímž uplynutí je snímek zobrazen na zobrazovacím tlačítku. V případě, že je tlačítko zmáčknuto, dojde ke spuštění čekací smyčky, která se ukončí dalším zmáčknutím tlačítka. Tím lze přehrávání kdykoliv pozastavit. Při ukončení čtení snímků je vyslán signál `void playFinished()`, čímž je zahájeno korektní uvolňování vlákna a použitých prostředků.

3.3 Výsledná aplikace

Výsledkem této práce je vytvořená aplikace, která byla pojmenována FFmpegGUI. Body specifikované zadáním byly splněny. Aplikace umožňuje sestavení grafu filtrů, přehrávání náhledu zpracovaného videa a zadávání parametrů jednotlivým filtrům. Kromě toho umožňuje automaticky propojit filtry na základě jejich pozic, uložit a načíst jak graf filtrů, tak i zpracované video. Nechybí ani dynamická inspekce vlastností propojení mezi filtry. Ukázka výsledné aplikace je na obrázku 3.3. V pravé části je seznam filtrů, vlevo seznam parametrů označeného filtru crop a tabulka s daty dynamické inspekce. Uprostřed pracovní plochy je jednoduchý graf filtrů a jeho náhled.



Obrázek 3.3: Ukázka výsledné aplikace.

3.4 Porovnání s existujícími řešeními

Při vytváření výsledné aplikace sloužily nástroje Graph-Edit a Gst-Editor jako hlavní zdroj inspirace. Při seznamování s těmito nástroji se mi velmi často stávalo, že docházelo k jejich neočekávanému pádu. Hlavní nedostatek, který podle mého názoru tato uživatelská rozhraní obsahují, je nepříliš uživatelsky vstřícné ovládání. Filtry je nutno přidávat pomocí dvojkliku a jsou umístěny na předem zvolené pozice. To znamená, že kvůli přehlednosti si je uživatel musí přemisťovat. Dále je seznam filtrů buď v dialogovém okně nebo v postranním panelu, který nelze žádným způsobem přesunout. Na druhou stranu nástroj Gst-Editor rozděluje filtry do kategorií podle společných vlastností a tím zajišťuje větší přehlednost a rychlejší vyhledávání. Celkové srovnání klíčových vlastnosti je demonstrováno tabulkou 3.4.

	FFmpegGUI	Gst-Editor	Graph-Edit
Vložení filtru	drag&drop	dvojklik	dvojklik
Spojení filtru	klávesová zkratka, nebo zmáčknutí ikony	tažení myší	tažení myší
Seznam filtrů	v hlavním okně	v hlavním okně	v dialogovém okně
Editor vlastností	v hlavním okně	v dialogovém okně	v dialogovém okně
Podpora náhledu	ano	ano	ano
Dynamická inspekce vlastností grafu	ano	ano	ne
Uložení/načtení	ano (XML formát)	ano (XML formát)	ano (XML formát a vlastní formát)
Nápověda pro každý filtr a parameter	ano	ne	ne
Vytvoření textové reprezentace grafu	ano	ano	ne
Podporované datové toky	video	audio/video	audio/video
Počet vstupů a výstupů	1	neomezeně	neomezeně

Tabulka 3.4: Porovnání klíčových vlastností výsledné aplikace s existujícími řešeními.

Závěr

Cílem této práce bylo vytvořit aplikaci s grafickým uživatelským rozhraním pro tvorbu grafů filtrů FFmpeg. Zadání specifikovalo dvě podmínky pro splnění – umožnit přehrát náhled a měnit parametry filtrů. Ze začátku je ovšem nutno pochopit související teorii a inspirovat se již existujícími řešeními pro jiné frameworky. To bylo uživateli představeno ve druhé kapitole této práce. Třetí kapitola analyzuje návrh a popisuje způsob implementace. Srovnání výsledku s podobnými aplikacemi je ve čtvrté kapitole.

Vývoj aplikace proběhl v operačním systému Ubuntu 14.04. Volba grafického toolkitu byla ovlivněna jeho multiplatformností a osobními zkušenostmi. Proto se Qt framework jevil jako ideální volba. Během implementace nebyly použity žádné platformně závislé prostředky, proto by měla být aplikace přeložitelná a spustitelná na všech podporovaných Qt a FFmpeg systémech. Největším problémem, kterému jsem musel čelit, byla absence kvalitní literatury a srozumitelných příkladů popisujících práci s FFmpeg knihovnami. Nicméně, zadání bylo splněno, a navíc byla přidána dynamická inspekce protékajících dat, možnost uložení a načtení grafů, a to ve formátu XML nebo ve tvaru textového řetězce pro nástroj ffplay. Zápis do souborů byl rovněž implementován, ale obsahuje drobné nedostatky, které potřebují doladit. Dále byly implementovány některé uživatelské pomůcky nad rámec zadání - nápověda k filtrům a parametrům, ošetření uživatelských vstupů, automatické propojení filtrů, možnost pozastavení přehrávání, atd. Výsledná aplikace byla uvolněna pod LGPL v2.1 licencí na serveru GitHub (dostupné z: https://github.com/RomanSichkaruk/FFmpegGUI).

Co se týče možných směrů dalšího vývoje, patří mezi ně podpora více vstupů a výstupů, podpora audia, možnost přehrávání více náhledů zároveň. Přehrávač náhledu umožňuje pozastavení přehrávání, ale nepodporuje přetáčení, zrychlení, zpomalení. Plnohodnotný přehrávač by určitě zlepšil celkový dojem z aplikace.

Literatura

[1] Bařina, D.; Zemčík, P.: *Multimédia: Studijní opora.* 2013.

[2] FFmpeg: *FFmpeg.* [Online; navštíveno 15.01.2017].
URL http://www.ffmpeg.org

[3] GStreamer: *GStreamer documentation.* [Online; navštíveno 15.01.2017].
URL https://gstreamer.freedesktop.org/documentation/

[4] Microsoft: *About DirectShow Filters.* [Online; navštíveno 15.01.2017].
URL
https://msdn.microsoft.com/en-us/library/windows/desktop/ms696274.aspx

[5] Microsoft: *About Media Foundation.* [Online; navštíveno 15.01.2017].
URL
https://msdn.microsoft.com/en-us/library/windows/desktop/ms696274.aspx

[6] Microsoft: *Video for Windows.* [Online; navštíveno 15.01.2017].
URL https://msdn.microsoft.com/en-us/library/windows/desktop/dd757708(v=vs.85).aspx

[7] The Qt company: *About Qt.* [Online; navštíveno 15.01.2017].
URL https://wiki.qt.io/About_Qt

[8] The Qt company: *All classes.* [Online; navštíveno 15.01.2017].
URL http://doc.qt.io/qt-5/classes.html

[9] Watkinson, J.: *Introduction to Digital Video.* Taylor & Francis, 2012, ISBN 9781136027611.
URL https://books.google.cz/books?id=0QAETdU_B5EC

Přílohy

Příloha A

Obsah CD

- **report.pdf** – tato technická zpráva
- **latex/** – složka se zdrojovými kódy pro LaTeX
- **src/** – složka se zdrojovými soubory aplikace
- **icons/** – složka s ikonami a obrázky
- **images.qrc** – soubor s použitými obrázky
- **configure** – konfigurační skript
- **README** – soubor s popisem aplikace a návodem na sestavení a spuštění
- **LICENSE** – licenční soubor
- **Excel@FIT/** – složka se soubory z konference Excel@FIT 2017. Obsahuje:
  poster.pdf – plakát z konference
  article.pdf – článek z konference
- **video/** – složka s video soubory. Obsahuje:
  clock.avi – základní testovací video
  howto.mkv – video ukazující práci s aplikací