VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

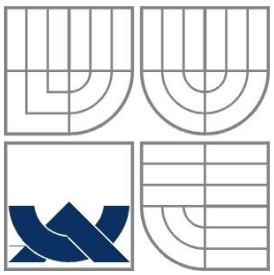
EDITOR VIDEA PRO PLATFORMU ANDROID

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

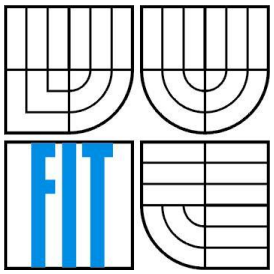
AUTOR PRÁCE
AUTHOR

BC. MAREK VYORAL

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

EDITOR VIDEA PRO PLATFORMU ANDROID

VIDEO EDITOR FOR ANDRIOD PLATFORM

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR

BC. MAREK VYORAL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. ALEŠ LÁNÍK

BRNO 2011

Zadání diplomové práce/12858/2010/xvyora03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Vyoral Marek, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Editor videa pro platformu Android**
Video Editor For Android Platform
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte metody zpracování videa a současné možnosti střihu videa. Seznamte se s mobilní platformou Android a vývojem nativního kódu na této platformě.
2. Navrhněte jednoduchý systém pro zpracování a střih videa s ohledem na rozšiřitelnost, obecnost a jednoduchost použití na dotykovém display.
3. Navržený systém implementujte pro platformu Android. Proveďte dostatečné otestování vaší aplikace.
4. Zhodnoťte dosažené výsledky a porovnejte je s dalšími existujícími programy.
5. Projekt presentujte plakátkem a demonstračním videem.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% 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: **Láník Aleš, Ing.**, UPGM FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

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

Cílem této práce je implementovat jednoduchou aplikaci umožňující editaci videa na platformě Android. V teoretické části jsou popsány současné možnosti střihu videa na počítači a zmíněny také dostupné aplikace pro Android. Dále práce popisuje samotnou platformu a její vlastnosti se zaměřením na vývoj aplikací, zejména těch nativních. Následně je popsána knihovna FFmpeg pro zpracování multimediálních formátů, která byla při implementaci použita. V další části se práce zabývá vytvářenou aplikací a popisuje její návrh, strukturu, uživatelské rozhraní a postup implementace. Následně práce předkládá výsledky výkonnostních testů při zpracování videa a jejich zhodnocení. V závěru práce je aplikace porovnána s podobnými již existujícími aplikacemi.

Abstract

Main goal of this thesis is implementation of a simple video editing application for Android platform. In the theoretical part are described present-day possibilities of video editing on computers and mentioned also existing applications for Android. Then the platform itself and its features are described with focus to development of applications, especially native ones. The FFmpeg library for multimedia processing is described next. It was used in implementation of application. The next part of thesis considers with implemented application and describes its design, structure, user interface and process of implementation. Then the results of video processing performance tests and their evaluation are presented. In the end is the application compared with similar existing applications.

Klíčová slova

Android, video editor, FFmpeg, libavcodec, libavformat.

Keywords

Android, video editor, FFmpeg, libavcodec, libavformat.

Citace

Vyoral Marek: Editor videa pro platformu Android, diplomová práce, Brno, FIT VUT v Brně, 2011

Editor videa pro platformu Android

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Aleše Láníka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Marek Vyoral
21. 5. 2011

Poděkování

Tímto děkuji svému vedoucímu diplomové práce Ing. Aleši Láníkovi za poskytnuté informace a bezproblémovou spolupráci.

Obsah

1 Úvod

Rostoucí obliba a široká nabídka takzvaných chytrých mobilních telefonů rozšířila tato zařízení mezi běžné uživatele. Chytré telefony umožňují uživatelům využívat nejen standardní funkce, které se od mobilních telefonů očekávají, jako uskutečňování hovorů, odesílání SMS zpráv a ukládání kontaktů, ale stále více umožňují využívat funkcionalitu podobnou, jakou mají desktopové počítače – instalace nových aplikací, plnohodnotné prohlížení webových stránek, připojení do WI-FI sítí, rychlé datové přenosy, přístup na sociální sítě, multimediální zábava, posílání emailů a podobně. Na rozdíl od klasických počítačů jsou navíc v poslední době standardem mezi chytrými telefony také technologie typu GPS, akcelerometry, různé typy senzorů, dotykové ovládání atd.

Také výkonnost takovýchto telefonů se stále rychleji zvyšuje – chytré telefony mají relativně výkonné procesory, RAM paměť v řádech stovek megabytů, datová úložiště v řádech gigabytů a také stále častěji hardwarovou akceleraci grafiky a multimediálních operací. Na takovémto hardwaru potom může běžet operační systém, který kromě standardních vestavěných aplikací umožňuje uživatelům libovolně instalovat nové aplikace a více či méně upravovat nebo měnit součásti samotného systému.

Právě jedním z takovýchto systémů je Android, pro který bude určena aplikace implementovaná jako součást této diplomové práce. Tato platforma byla zvolena pro její otevřenost, značné rozšíření, její velmi slibné vyhlídky do budoucna, její dobrou dokumentaci a dostupnost tutoriálů a ukázkových příkladů.

Po implementaci aplikace bude možné vyzkoušet možnosti této platformy v oblasti zpracování videa. Výkonnost mobilních procesorů neustále narůstá, taktéž velikost RAM paměti a úložného prostoru. V současné době by tedy tyto parametry měly být již dostatečné pro alespoň jednoduché zpracování videa. Každopádně výkon v budoucnu neustále nadále poroste, takže i pokud výkonnost současných zařízení nebude zcela dostatečná, určitě je jen otázka času, kdy se tak stane.

V poslední době se také začínají na trhu objevovat různé tablety, které kromě výkonného hardwaru mají taktéž dostatečně velké displeje pro pohodlnou práci pomocí dotykového ovládání. Operační systém Android se na těchto tabletech často objevuje, proto jsou tato zařízení slibným prostorem pro využití aplikací pro editaci videa.

Praktickou částí této diplomové práce je vytvoření jednoduché aplikace pro editaci videa. Záměrem je implementovat video editor umožňující pracovat s videi v různých formátech, umožnit vystřižení pouze požadované části, umožnit spojovat více videí a vystřižených části dohromady a poskládat tak úplně nové video. To pak bude možné exportovat do různých video formátů s možností nastavení výstupních parametrů. Další možností úpravy bude vkládání obrazových efektů. Editor také bude možné využít pro překódování existujících videí do jiných formátů. Aplikace bude zaměřena hlavně na mobilní telefony, proto bude její rozhraní přizpůsobeno malým mobilním displejům a orientováno na jednoduchost ovládání pomocí prstů.

Po implementaci takovéto aplikace bude potom zajímavé sledovat jak rychle jsou současná mobilní zařízení schopna pracovat s videem, proto bude také provedeno testování a jeho výsledky budou dále analyzovány.

2 Možnosti editace videa na mobilních zařízeních

Tato kapitola poskytuje základní teorii nutnou pro zasazení problematiky do širšího kontextu. Nejprve se obecně zabývá tím, jaké jsou současné možnosti zpracování videa. Poté jsou poskytnuty základní informace o samotné platformě Android a způsobu vývoje aplikací na této platformě. V závěru kapitoly jsou popsány vlastnosti a důležité části knihovny FFmpeg, která bude při implementaci video editoru využita.

2.1 Zpracování a střih videa

Tato podkapitola poskytuje přehled současných možností a vybraných aplikací pro zpracování videa. Základním cílem je ukázat existující dostupné aplikace na platformě Android. Vzhledem k tomu, že takovýchto aplikací mnoho není, budou nejprve zmíněny také aplikace určené pro klasické desktopové počítače. Ty totiž také mohou posloužit jako inspirace pro implementační část této práce. Jednak svou funkcionalitou, ale také způsobem práce s videem nebo uživatelským rozhraním.

2.1.1 Editory pro desktopové systémy

Editorů videa existuje velké množství, v širokém spektru jak z hlediska ceny, tak z hlediska funkcionality. Přehled v této oblasti obsahuje webová stránka [1]. Pro větší přehlednost editory rozdělme čtyř kategorií.

Profesionální editory

Do této kategorie spadají zástupci velkých komplexních nástrojů, které mohou být použity také v profesionální oblasti. Sem patří např. *Adobe Premiere Pro*, *Avid Media Composer*, *Sony Vegas* nebo *Final Cut*. Tyto aplikace jsou často používány také ve filmovém průmyslu. Představují špičku mezi softwarem pro zpracování videa a tomu odpovídá také jejich cena (i více než $1000).

Jako příklad softwaru z této oblasti byl vybrán *Adobe Premiere Pro* [2]. Je součástí balíčku *Creative Suite* pro zpracování videa a obrazu, nicméně lze ho zakoupit i samostatně. K dispozici existuje také odlehčená jednodušší verze, zaměřená na domácí zpracování videa *Adobe Premiere Elements*. Existuje ve verzích pro Windows a pro Mac OS. Editor umožňuje nelineární střih videa v reálném čase. Podporuje video ve vysokém rozlišení (až 10240×8192 pixelů) s 32bitovou hloubkou barev a editaci vícekanálového prostorového zvuku. Po doinstalování odpovídajícího plug-in modulu podporuje také střih a export 3D videa. Díky vlastnosti *real-time rendering* je možné zobrazovat okamžitý náhled výsledného videa. Teoreticky umožňuje využívat při editaci neomezený počet video a audio stop (to je samozřejmě omezeno hardwarovými prostředky).

Podporuje širokou škálu vstupních a výstupních formátů, video efektů a různého hardwaru (všemožné karty pro editaci videa, zachytávání videa, hardwarovou akceleraci apod.). Pomocí plug-in modulů lze rozšířit funkcionalitu, přidat video efekty, filtry a podporu dalších formátů a kodeků.

Obrázek 1. Adobe Premiere Pro.

Placené video editory pro domácí využití

Tato skupina je velmi rozsáhlá a nabízí širokou škálu editorů od různých výrobců v různých cenových hladinách (obvykle v řádu desítek až několika stovek dolarů). Nejrozšířenějšími zástupci této kategorie jsou *Adobe Premiere Elements*, *CyberLink PowerDirector*, *Nero Vision*, *Pinnacle Studio*, *Ulead VideoStudio* nebo *MAGIX Movie Edit*.

Editory tohoto typu poskytují dostatečnou funkcionalitu pro domácí použití, ale mnohé z nich i v oblasti poloprofesionálního zpracování videa. Principy střihu jsou prakticky podobné jako u profesionálních editorů pomocí několika audio a video stop. Tyto nástroje disponují obvykle širokou podporou formátů, množstvím filtrů a efektů a různých plug-in modulů rozšiřujících funkcionalitu.

Jednodušší komerční video editory

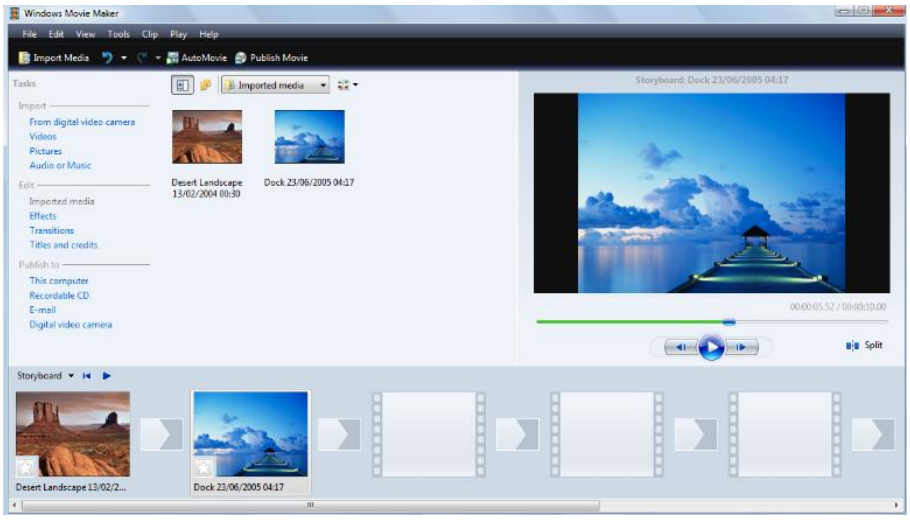
Nástroje z této kategorie jsou obvykle různé proprietární aplikace, které jsou obvykle zdarma nebo distribuovány jako součásti operačních systémů, případně jako jejich možné rozšíření. Do kategorie těchto nástrojů můžeme zařadit aplikace typu *Windows Movie Maker*, *Apple iMovie*, *Pinnacle Video Spin*, *Lightworks* a podobně.

Vybraným zástupcem z této kategorie je *Windows Movie Maker* [3] (aktuálně Windows Live Movie Maker jako součást balíčku Windows Live Essentials). Je jednoduchým editorem videa pro operační systémy Windows, dostupný již od verze *Windows Me*. Poskytuje přehledné a intuitivní uživatelské rozhraní. Obsahuje galerii, do které je možné importovat videoklipy, které lze následně pomocí střihů rozdělovat na další klipy. Z videoklipů v galerii je pak možné sestavit výsledné video poskládáním videoklipů do jakési časové osy. Tento jednoduchý a přímočarý přístup ke střihu posloužil k inspiraci při implementaci video editoru pro Android. Tento přístup se zdá být vhodný pro malé displeje telefonů a mobilních zařízení, kde by mohla být nějaká vícestopá časová osa příliš složitá a nepřehledná.

Windows Movie Maker dále podporuje vkládání různých efektů a přechodů. Umožňuje vkládat speciální sekvence, jako závěrečné nebo koncové titulky. Také je možné do galerie importovat

zvukové soubory a ty pak přidávat do výsledného videa pomocí umístění do audio stopy. Také je zde možnost zaznamenat mluvený komentář k celému výslednému videu. Od novějších verzí podporuje publikování videa na všemožné webové služby (Youtube, Facebook apod.).

Nevýhodou tohoto editoru je slabá podpora formátů, ze kterých lze videoklipy importovat. Například MP4, 3GP, FLV nebo SWF nejsou podporovány, ani když jsou v systému odpovídající dekodéry nainstalovány. Export pak lze provádět pouze do formátů Windows Media (WMV video, WMA audio) nebo DV/AVI.

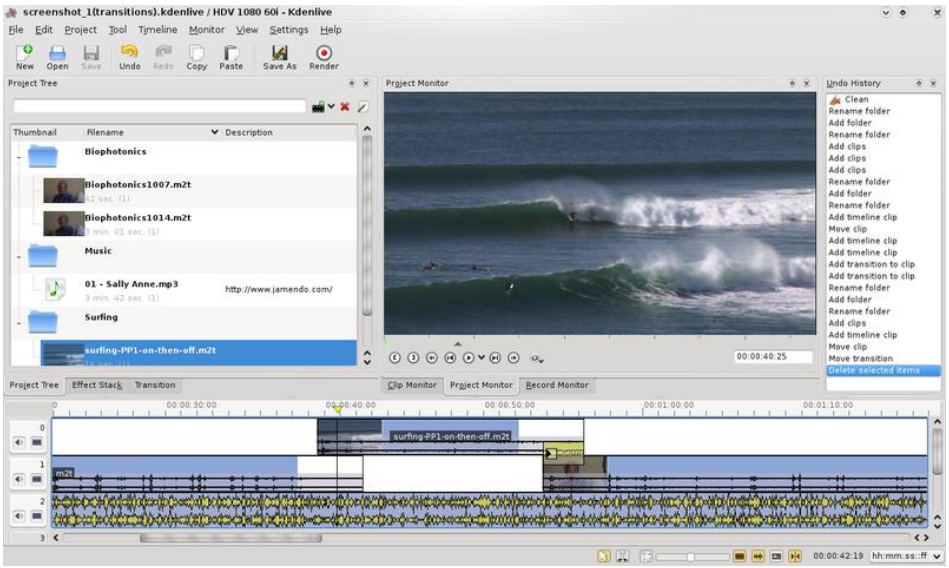


Obrázek 2. Windows Movie Maker.

Open-source video editory

Zatímco většina softwaru spadajícího do předchozích třech kategorií je určena pro systémy Windows nebo Mac OS, v kategorii open-source video editorů (dostupných obvykle pod licencí GPL) jasně dominuje Linux. Takových projektů existuje celá řada, mezi nejznámější patří *Kdenlive*, *Kino*, *Cinelerra*, *OpenShot Video Editor* nebo *Video Sequence Editor* (plug-in pro *Blender*, k dispozici i pro Windows a Mac OS). Výhodou těchto editorů kromě nulové ceny bývá také dostupnost zdrojových souborů, široká komunita uživatelů a obvykle i vývojářů. Na druhou stranu nemusí být jejich funkcionalita vždy srovnatelná s rozsáhlými komerčními video editory.

Jako příklad editorů z této skupiny byl vybrán editor *Kdenlive* [4], který je zaměřený na jednoduchost ovládání a flexibilitu. Je možné jej zprovoznit jednak na Linuxových operačních systémech, ale také na Free BSD nebo Mac OS X. Je postavený na frameworku MLT (Media Lovin´ Toolkit). Díky použití knihovny FFmpeg podporuje širokou škálu multimediálních formátů a kodeků. Editor využívá prostředí KDE, které proto nezbytně potřebuje ke svému provozu. Podobně jako většina ostatních editorů umožňuje práci s několika video a audio stopami a nabízí širokou škálu různých efektů. Pro zpracování efektů používá různé knihovny, například dříve zmiňovanou MLT, Frei0r, SoX nebo LADSPA.



Obrázek 3. Kdenlive.

2.1.2 Aplikace pro Android

Hlavním zdrojem pro získání aplikací pro tuto platformu je rozhodně Android Market [5]. Ten umožňuje jednoduše distribuovat aplikace koncovým uživatelům. Vývojáři mohou své aplikace snadno publikovat (po zaplacení poplatku za registraci vývojáře) a pomocí Marketu jsou tak instalační balíčky k dispozici prakticky všem Android zařízením ve většině částí světa. Odpadá tak složitá distribuce balíčků na straně vývojářů. Velmi snadno lze také publikovat updaty a nové verze, které se opět velmi rychle dostanou k uživatelům aplikací. Také lze získávat hodnocení od uživatelů, připomínky nebo hlášení o chybách v aplikaci. Navíc je zde možnost publikovat aplikace nejenom zadarmo, ale aplikace taktéž zpoplatnit, což je opět velice výhodné.

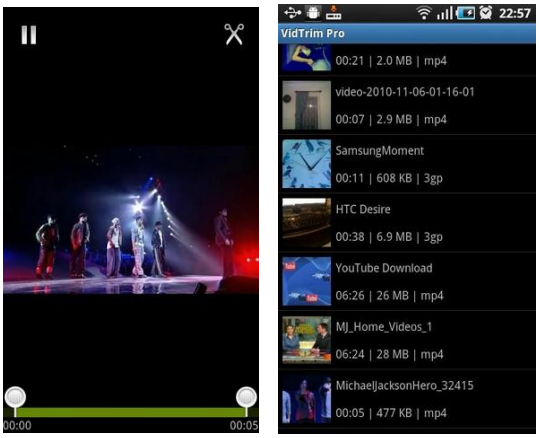
Díky zmíněným vlastnostem je tento distribuční kanál mezi vývojáři velmi oblíben. Drtivá většina aplikací, které pro Android vznikají, jsou proto dostupné právě zde. Bohužel existují i negativa, mnohdy je nemožné oficiálně (a legálně) získat instalační balíček jinou cestou než přes Market. U mnoha projektů, které jsou třeba i zdarma, není mnohdy možné stáhnout instalační balíček např. z oficiálních stránek projektu a jedinou možností zůstává pouze Android Market. Z výše zmíněných důvodů lze předpokládat, že Android Market je nejlepším zdrojem pro hledání již existujících video editorů. Každopádně hledání aplikací pro editaci videa nebylo omezeno jen na Market, ale také pro jistotu rozšířeno na internet a další alternativní zdroje aplikací (např. AppBrain [6]).

VidTrim Video Trimmer

Tato jednoduchá aplikace dostupná z Android Marketu existuje ve dvou verzích – zdarma a placená. Verze zdarma umožňuje otevřít video soubor, přehrávat jeho obsah (bez zvuku) a označit počáteční a koncový čas střihu. Následně je možné videoklip oříznou v daných místech a zvolit, zda přepsat původní soubor nebo vytvořit nový. Aplikace využívá knihovnu FFmpeg a žádné kódování videa neprovádí, pouze odstraňuje datové pakety formátu před a po zvolených časech (de facto pouze práce s obsahem souboru). Proto je provedené ořezání velice rychlé a pro mobilní zařízení výpočetně

nenáročné. Tato verze zdarma spadá do kategorie 50 000 až 250 000 stažení a má relativně dobré hodnocení uživatelů.

Placená verze (aktuální cena $2,57) nabízí další funkcionalitu a odstraňuje reklamu, která je ve verzi zdarma přítomna. Dále umožňuje exportovat zvolené snímky z videa jako obrázky a umí překódovat videoklip do menšího rozlišení obrazu. Nicméně výsledným formátem je vždy MP4 bez možnosti jiné volby. Aplikaci lze také využít jako jednoduchý manažer video souborů, který umožňuje jejich přejmenování a odesílání přes e-mail nebo Bluetooth. Placená verze spadá do kategorie 1 000 - 5 000 stažení. V hodnocení uživatelů se často objevuje kritika nefunkčního překódování a další různé problémy s ním spojené.



Obrázek 4. Aplikace VidTrim.

Snip Video Trimmer

Další jednoduchá aplikace pro ořezávání videoklipů. Je dostupná na Android Marketu, ale jenom v placené verzi (současná cena je $0,99). Aplikace umožňuje podobně jako dříve zmíněný VidTrim nastavit počáteční a koncový čas a videoklip oříznout. Také podporuje různé způsoby odesílání výsledku. Pracuje s formáty 3GP, MP4 a všemi ostatním, které Android standardně podporuje. Použití nějaké externí multimediální knihovny typu FFmpeg není nikde uvedeno, proto se zdá, že aplikace využívá pouze standardního Android API pro práci s multimediálními formáty.

Aplikace spadá do kategorie 1 000 až 5 000 stažení. Hodnocení uživatelů je velmi kritické, stěžují si především na nefunkčnost a časté pády.

Obrázek 5. Aplikace Snip Video Trimmer.

Clesh video editor

Aplikace je dostupná z Android Marketu a existuje pouze v placené verzi (současná cena $4,02). Nabízí sdílení fotografií a videoklipů na všemožných internetových umístěních. Umožňuje střih videoklipů a jejich spojování do jednoho výsledného videa. Podporuje vkládání 18 různých přechodů mezi jednotlivými videoklipy a vytváření obrázků z vybraných snímků videa. Umožňuje také zaznamenávat video z vestavěného fotoaparátu a dále jej upravovat.

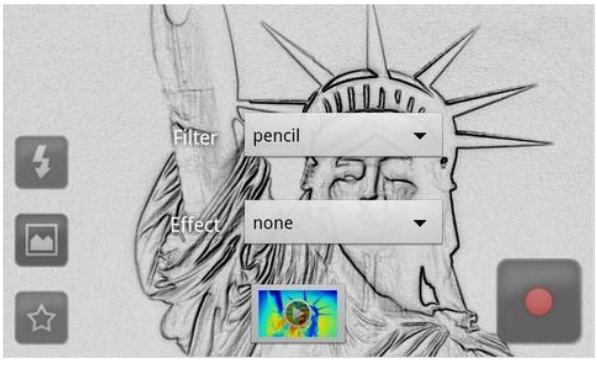
Tato aplikace je ovšem pouze tenkým klientem cloud-based video editoru Clesh. Takže veškerý multimediální obsah je odesílán na vzdálený server, který provádí výpočetní operace a kódování a výsledky pak zpětně odesílá na mobilní zařízení. Toto řešení je rozhodně velmi rychlé a šetrné k baterii zařízení. Nicméně klade vysoké nároky na datové přenosy, takže použití tohoto editoru je do jisté míry omezeno.

Aplikace je relativně nová, proto zatím spadá do kategorie 100 až 500 stažení. V hodnocení uživatelů se objevují jak velmi pozitivní ohlasy, tak ty velmi negativní (nefunkčnost na různých zařízeních, velmi dlouhá doba odesílání na server, pády aplikace, nepohodlné ovládání).

Obrázek 6. Clesh video editor.

Videocam Illusion

Tato aplikace není editorem ve smyslu střihu nebo spojování různých videoklipů, nicméně provádí úpravu videa, proto ji lze také za jistý typ editoru považovat. Umožňuje pořizovat videosnímky z kamery zařízení a v reálném čase aplikovat různé obrazové filtry a efekty. Existuje jak ve verzi zdarma (50 000 - 250 000 stažení), tak v placené (současná cena $2,75, 1 000 až 5 000 stažení aplikace), která přináší další filtry a efekty. Podobných aplikací existuje ještě několik, ovšem jejich počet stažení i hodnocení uživatelů je nižší a není třeba je zde dále zmiňovat.



Obrázek 7. Videocam Illusion.

2.2 Platforma Android

Tato kapitola nejprve seznámí čtenáře se stručnou historií platformy. Poté poskytne přehled o struktuře operačního systému, což je pro vývoj aplikací velmi důležité. Následuje popis balíčků SDK a NDK nutných pro vývoj aplikací, resp. nativních aplikací. Další podkapitola se poté zaměřuje na vývoj nativních aplikací a strukturu Android projektu.

2.2.1 Historie

Za vývojem platformy Android [7] [8] stojí americká společnost *Google*. Tato společnost působící do té doby převážně v oblasti internetových aplikací a vyhledávání, se v roce 2005 rozhodla vstoupit do oblasti mobilních zařízení. V témže roce koupila malou, téměř neznámou společnost *Android Inc.*, která se zabývala vývojem mobilních aplikací pro tzv. chytré telefony. Ve stejné době probíhal raketový start tehdy nového mobilního zařízení iPhone konkurenční společnosti *Apple*. Stoupající obliby a prodeje chytrých multimediálních telefonů se rozhodl *Google* využít a přijít s vlastním zařízením. Mimo to byl odstartován vývoj zcela nového open-source sytému, který by tak byl konkurencí do té doby existujícímu *Symbianu, Apple iOS* a *Windows Mobile*.

Vývoj [9] tohoto nového systému nebyl v počátcích oficiálně veřejně ohlášen, proto se objevovaly nejdříve pouze spekulace a nepotvrzené informace o prototypech telefonů a jednáních s výrobci a mobilními operátory. Přesto nebylo zcela jasné, zdali *Google* skutečně hodlá na trh s mobilními zařízeními vstoupit. V září roku 2007 si nechala společnost registrovat několik patentů v této oblasti, což její úmysly víceméně nepřímo potvrdilo.

Nicméně na vývoji sytému Android se nepodílí pouze firma *Google*, ale uskupení *Open Handset Alliance* (OHA), složené z mnoha společností, například *Texas Instruments, Broadcom Corporation, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel* a *T-Mobile*. Uskupení bylo založeno 5. listopadu 2007 s cílem vyvíjet otevřené standardy v oblasti mobilních zařízení. Uskupení OHA se dne 9. září 2008 rozšířilo o nové členy, mezi nimiž jsou například společnosti *PacketVideo, ARM Holdings, Atheros Communications, Asustek Computer, Garmin, Softbank, Sony Ericsson, Toshiba Corporation* nebo *Vodafone Group*.

Již 23. září 2008 byla uveřejněna první verze systému Android 1.0. Další verze 1.1 přinášející především opravy chyb byla zveřejněna 9. února 2009. Vývoj systému dále pokračoval a relativně rychle se objevovaly nové verze až do současné aktuální verze 2.3.4.

S nástupem a bouřlivým rozvojem tabletů bylo potřeba tuto situaci zohlednit, proto vznikla samostatná verze 3.0 určena právě pro tablety, uzpůsobená jejich potřebám (velikost displejů, absence GSM modulů, nástup hardwarové akcelerace grafiky nebo nových platforem jako nVidia Tegra apod.).

Rychlý nástup nových verzí rozhodně přináší pochopitelné výhody, nicméně na druhou stranu z pohledu vývojáře aplikací také značné nevýhody. Na trhu existují vedle sebe zařízení s různými verzemi systému a to zcela jistě může přinášet problémy s kompatibilitou aplikace (nehledě na velmi rozmanitou škálu hardwaru).

2.2.2 Struktura

Základem systému Android je **Linuxové jádro** verze 2.6 poskytující základní funkcionalitu jako správu paměti, správu procesů, obsluhu sítě nebo ovladače a je nejnižším rozhraním mezi hardwarem mobilního zařízení a vyššími vrstvami systému [10].

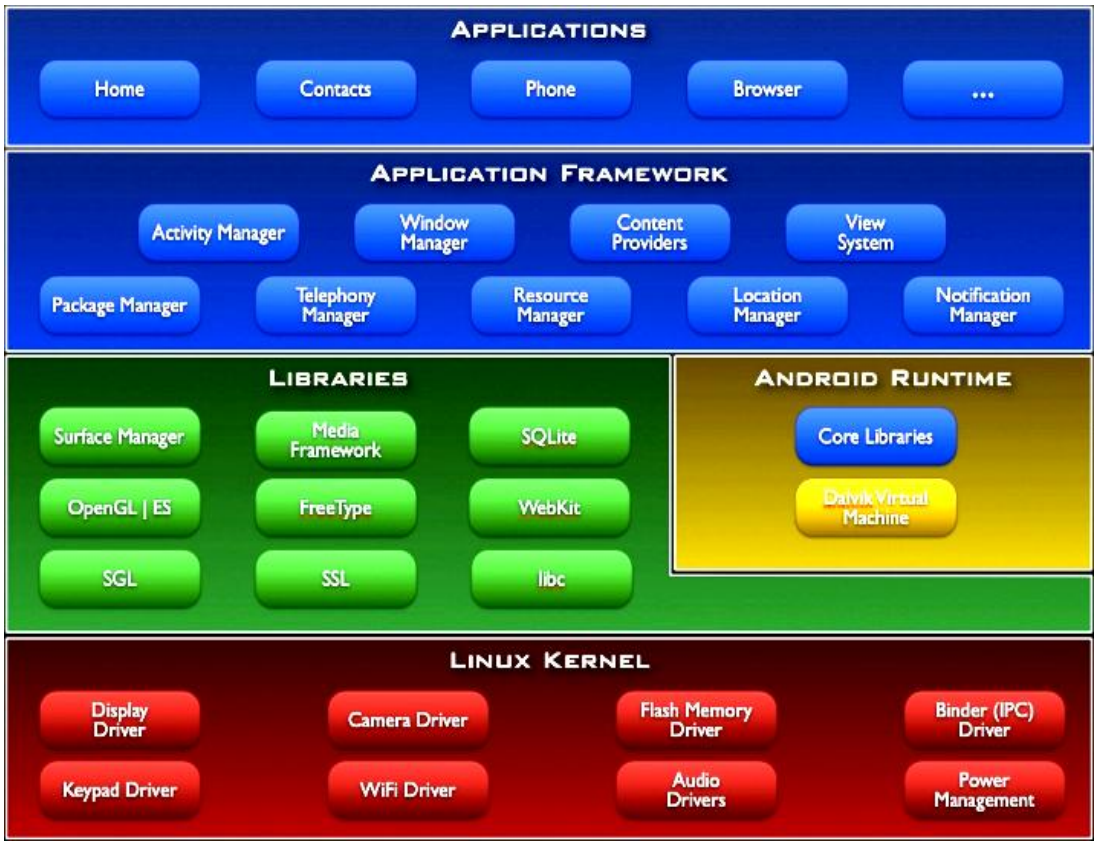
Vrstva **Android runtime** obsahuje základní knihovny pro běh aplikací. Každá aplikace běží ve svém vlastním procesu na svém vlastním virtuálním stroji (Dalvik Virtual Machine). Takových virtuálních strojů pochopitelně může běžet více zároveň. Aplikace jsou napsány v jazyce Java, poté přeloženy do formátu *.dex* (Dalvik Executable), což je kód zpracovávaný virtuálním strojem. Práce s vlákny a nízkoúrovňová správa paměti není řešena virtuálním stojem, ale přímo Linuxovým jádrem.

Vrstva knihoven obsahuje C/C++ knihovny. Knihovna *System C* je implementací základní knihovny jazyka C *libc* přizpůsobena potřebám mobilních zařízení. Dále jsou zde obsaženy *Media Libraries* postaveny na OpenCORE knihovnách od PacketVideo pro práci s video a audio formáty a obrázky: MPEG4, H.264, MP3, AAC, AMR, JPG, a PNG. Knihovna *Surface Manager* poskytuje základní rozhraní pro přístup k displeji. Pro 2D vykreslování využívá knihovnu *SGL* . Pro 3D grafiku jsou k dispozici *3D libraries* postavené na OpenGL ES. Podporují hardwarovou akceleraci, pokud je na daném zařízení dostupná. V případě že ne, je k dispozici vysoce optimalizované softwarové 3D vykreslování. Pro vykreslování fontů slouží knihovna *FreeType*, která umožňuje vykreslovat jak bitmapové, tak vektorové fonty. Pro ukládání dat je k dispozici knihovna *SQLite*, poskytující lightweight databázi. Rychlý a optimalizovaný engine pro webové prohlížeče obsahuje knihovna *LibWebCore*.

Aplikační Framework poskytuje vývojářům přístup k funkcím systému. Cílem bylo vytvořit rozhraní tak, aby bylo snadné, bezpečné a svobodné přistupovat ke všem možnostem, které systém nabízí a tvořit tak rozmanité aplikace. Dále je kladen důraz na znovupoužitelnost kódu a sdílení, což umožňuje aplikacím poskytovat si vzájemně funkcionalitu (za dodržení jistých bezpečnostních omezení). Tato vrstva obsahuje například:

- *Views* - slouží jako stavební prvky uživatelského rozhraní – okna, tlačítka, seznamy, menu posuvníky apod.
- *Content Providers* – umožňují aplikacím sdílet data mezi sebou.
- *Resource Manager* – poskytují aplikacím přístup ke zdrojům, které jsou mimo samotný kód aplikace, jako obrázky, lokalizované textové řetězce, XML soubory popisující uživatelské rozhraní atd.
- *Notification Manager* – umožňuje aplikacím zobrazovat upozornění ve stavovém řádku systému.
- *Activity Manager* – spravuje životní cyklus aplikací (spuštění, uspání, probuzení, ukončení).

Vrstva aplikací obsahuje standardní aplikace distribuované se systémem, jako webový prohlížeč, obsluhu hovorů, správu kontaktů, domovskou plochu, e-mailový klient a podobně. Tyto aplikace lze díky modularitě systému nahradit a samozřejmostí je zde i možnost instalace mnoha dalších uživatelských aplikací.



Obrázek 8. Struktura operačního systému Android [10].

2.2.3 SDK

Balíček Android SDK [11] (Standard Development Kit) obsahuje nástroje a API potřebné k vyvíjení aplikací, debugger, knihovny, ukázkové příklady aplikací, dokumentaci, tutoriály a emulátor mobilního telefonu pro testování aplikací. Součásti SDK jsou rozděleny na jednotlivé komponenty, které lze instalovat pomocí nástroje *android*. Komponenty lze vybírat podle verzí systému, což umožňuje testovat aplikace na různých verzích platformy.

SDK je oficiálně k dispozici pro operační systémy Microsoft Windows, pro Mac OS a operační systémy typu Linux. Windows verzi lze provozovat jak na 32bitové architektuře, tak na 64bitové. Pro Mac OS existuje pouze 32bitová verze a oficiálně doporučeným Linuxovým systémem je *Ubuntu* verze *Lucid Lynx*.

Oficiálně podporovaným vývojovým prostředím je *Eclipse IDE* verze 3.4 a vyšší. Také je doporučeno mít pro snazší a komfortnější vývoj aplikací v tomto prostředí nainstalovaný *JDT plugin* (Java Development Tools). Vzhledem k tomu, že aplikace pro Android jsou psány v jazyce Java, je potřeba mít pro vývoj nainstalován balíček *JDK*, alespoň verze 5 (Java Development Kit). Je možné také použít jiná vývojová prostředí, jako například *NetBeans*, nicméně pluginy pro jiná IDE než *Eclipse* nejsou oficiálně podporovány a jejich vývoj je plně v rukou vývojářů třetích stran.

Alternativou je vývoj aplikací bez komplexního vývojového prostředí pomocí textového editoru a CLI nástrojů z balíčku *JDK*. V tomto případě je potřeba ještě použít nástroj *Apache Ant* alespoň verze 1.8.

Standard Development Kit obsahuje následující nástroje [11] [12]:

- **Android Development Tools Plugin** – rozšíření pro *Eclipse IDE* usnadňující vývoj v tomto prostředí (ladění, snadný překlad, spouštění, vytváření balíčků).
- **Android emulátor** – emulátor zařízení se systémem Android.
- **Android Virval Devices (AVD)** – instalace, správa a vytváření emulátorů. Počet emulátorů není omezen, takže je možné používat emulátory různých verzí systému a s různými "hardwarovými" vlastnostmi, jako je velikost paměti, rychlost sítě, rozlišení displeje a podobně.
- **Hierarchy Viewer** – umožňuje ladit a optimalizovat uživatelské rozhraní aplikací. Dokáže zobrazit výsledné rozložení prvků GUI na základě jejich definice v souborech XML. Umožňuje také zobrazit strom reprezentující hierarchii. Dále umí zobrazit GUI v režimu *Pixel Perfect View*, kde může vývojář pomocí přiblížení a mřížky doladit jemné grafické detaily na úrovni jednotlivých pixelů.
- **Draw 9-patch** – nástroj pro vytváření *NinePatch* grafiky stylem WYSIWYG. *NinePatch* je bitmapa rozdělená do 9 sekcí. Čtyři rohy jsou neměnné, 4 sekce mezi nimi jsou roztažitelné podle jedné osy a prostřední sekce je roztažitelná v obou osách. Toto lze výhodně využít pro velikostně proměnlivé grafické prvky, jako například tlačítka.
- **Dalvik Debug Monitor Service (ddms)** – tato služba umožňuje spravovat procesy na emulátorech nebo připojených fyzických zařízeních, zejména při ladění. Umožňuje ukončovat a ladit procesy, trasovat chyby, sledovat data na haldě, sbírat informace o vláknech, pořizovat snímky obrazovky a podobně.
- **Android Debug Bridge (adb)** – rozhraní pro ladění aplikací. Umí instalovat *.apk* balíčky do emulátorů nebo připojených fyzických zařízení a přistupovat k systému Android přes příkazový řádek.
- **Android Asset Packaging Tool (aapt)** – vytváření, prohlížení a modifikace výsledných distribučních balíčků obsahujících binární soubory pro Dalvik Virtual Machine a další zdroje jako grafika, XML soubory apod.
- **Android Interface Description Language (aidl)** – umožňuje vytvářet rozhraní pro komunikaci procesů pomocí jazyka IDL. Díky tomu si mohou rozdílné procesy a služby v systému předávat data pomocí meziprocesové komunikace (IPC).
- **sqlite3** – aplikace si uchovávají svá data v SQLite databázi. Pomocí tohoto nástroje je možné k této databázi přistupovat.
- **Traceview** – grafický prohlížeč aplikačních logů. Používá se pro ladění, optimalizaci a hledání chyb v aplikacích.
- **mksdcard** – umožňuje vytvořit obraz virtuální paměťové karty se souborovým systémem FAT32, která může být připojena k emulátoru a simulovat tak připojené externí paměťové úložiště.
- **dx** – překladač Java bytecode ze souborů *.class* do spustitelných souborů *.dex* (Dalvik Executable Format), které pak běží na Dalvik Virtual Machine.
- **UI/Application Exerciser Monkey** – nástroj pro stresové testování aplikace, především uživatelského rozhraní a stability. Pseudo-náhodně generuje události, vstupy uživatele, stisky kláves, kliknutí, systémové události a podobně.

- **monkeyrunner** – API pro kontrolu Android zařízení pomocí programů napsaných v jazyce Python. Programy napsané v tomto jazyce lze tak použít pro instalace a spouštění aplikací, posílání událostí, snímání obrazovky a obecně k testovacím účelům.
- **android** – skript pro správu virtuálních zařízení (emulátorů) a generování překladových souborů pro nástroj *Ant*. Poskytuje také správu, instalaci a update SDK komponent.
- **zipalign** – nástroj pro optimalizaci distribučních *.apk* balíčků. Zarovnává data pro rychlejší přístup.

2.2.4 NDK

Native Development Kit [13] je balíček nástrojů určených pro vývoj aplikací v nativním kódu, na rozdíl od SDK, které obsahuje nástroje pro vývoj aplikací v jazyce Java. NDK umožňuje implementovat části kódu aplikací v jazycích C nebo C++. Toto je výhodné především v částech aplikace, kde je požadována výkonnost a rychlé zpracování. Přece jenom je programovací jazyk Java jazykem interpretovaným, což přináší jistou výkonnostní nevýhodu oproti nativnímu kódu. Nicméně použití nativního kódu automaticky nepřináší vždy znatelné zrychlení. Použitím nativního kódu navíc narůstá komplexita programu a zvyšuje se riziko nepřehlednosti kódu. Proto je třeba velmi dobře zvážit, zdali je danou funkci vůbec potřeba implementovat v C/C++ nebo postačí standardní Java implementace. Vhodnými kandidáty na implementaci v nativním kódu jsou algoritmy intenzivně využívající procesor (kódování, zpracování signálů, složité výpočty…).

Další možností využití je použití již existujících zdrojových kódů v C/C++ a díky NDK je tak zprovoznit na platformě Android. Díky tomu je možné na Androidu provozovat aplikace, projekty a knihovny z jiných platforem (např. *OpenCV*, *FFmpeg*). Právě toho bude využito v implementační části práce. Konkrétně knihovna FFmpeg pro dekódování a kódování videa a zvuku. Knihovna je napsaná v jazyce C, což umožňuje její překlad a provoz v systému Android. Například známá aplikace *RockPlayer* pro přehrávání videa právě této knihovny taktéž využívá.

K provozu NDK vyžaduje nainstalované SDK minimálně verze 1.5. Stejně jako SDK existuje NDK pro systémy Windows, Mac OS X a Linux. Pro všechny zmíněné platformy je potřeba také *GNU Make* alespoň verze 3.81 a nástroj *GNU Awk* případně *Nawk*. Na platformě Windows je třeba mít nainstalován ještě *Cygwin* ve verzi alespoň 1.7.

NDK podporuje překlad do nativího kódu pro instrukční sady ARMv5TE, ARMv7-A a x86.

Obsah NDK

Native Development Kit [13] obsahuje API, ukázkové aplikace, dokumentaci a vývojové nástroje jako překladače z C/C++ do nativního kódu, linkery a nástroje pro snadnou kompilaci a začlenění nativních částí do projektu aplikace nebo *.apk* balíčku. Také obsahuje nativní API:

- *libc* – C knihovna.
- *libm* – matematická knihovna.
- JNI rozhraní.
- *libz* – Zlib komprese.
- *liblog* – logování.

- *OpenGL ES* – verze 1.1 a 2.0.
- *libjnigraphics* – přímý přístup do pixel bufferu.
- základní C++ knihovny.

2.2.5 Vývoj nativních aplikací

Pro vývoj aplikací v nativním kódu je potřeba kromě správně nainstalovaného balíčku nástrojů SDK také zprovoznění a konfigurace balíčku NDK.

Konfigurace NDK

NDK archiv lze stáhnout z oficiálních internetových stránek pro Android vývojáře [14]. Archiv není třeba nijak instalovat, stačí ho pouze rozbalit do libovolné složky. Jediným omezením je doporučení zvolit cestu, která neobsahuje žádné mezery.

V případě instalace na systémech Windows je třeba nainstalovat také prostředí *Cygwin* verze 1.7 nebo novější, které poskytuje základní nástroje z Linuxových operačních systémů (např. GNU Make). Ty jsou totiž NDK systémem využívány a bez nich není možné překlad provádět. Při instalaci *Cygwin* prostředí je vhodné k základním instalovaným balíčkům s nástroji přidat také skupinu nástrojů *Devel* (vývojové nástroje a utility). V prostředí Linux je třeba mít zmíněné vývojové nástroje také nainstalovány. Ve všech prostředích je poté vhodné nastavit proměnnou prostředí ANDROID_NDK_ROOT s cestou ke složce s NDK nástroji. Pokud je k vývoji používáno nějaké integrované vývojové prostředí, což je v případě vývoje Android aplikací nejspíše Eclipse, je vhodné také do tohoto IDE doinstalovat podporu pro C/C++ kód.

Ke spuštění překladu slouží skript *ndk-build* z balíčku NDK, který po spuštění ve složce projektu dané aplikace provede kompilaci zdrojových a hlavičkových souborů v jazyce C/C++, které musí být umístěny ve složce *jni*. Při kompilaci jsou vytvořeny ve složce aplikace další podložky *obj* (s přeloženými objektovými soubory) a *libs* (s výslednými nativními knihovnami .so). V operačních systémech Windows je potřeba tento skript spouštět z prostředí Cygwin, nelze jej úspěšně spustit z běžné systémové příkazové řádky.

Princip práce s nativním kódem

Aplikace na platformě Android jsou napsány v jazyce Java, zkompilovány do formátu *.dex* (Dalvik Executable) a jakožto interpretovaný jazyk spouštěny na virtuálních strojích (Dalvik Virtual Machine). Naproti tomu kód zkompilovaný pomocí NDK je přeložen přímo do nativního kódu cílového procesoru. K dispozici jsou poté nativní knihovny, které lze volat z Java kódu aplikace.

Existují dva způsoby vývoje nativního kódu [13]. Prvním z nich je klasické programování aplikace v jazyce Java s využitím standardního Android frameworku a přístup k funkcím v nativním kódu pomocí rozhraní JNI (Java Native Interface). Tento přístup lze použít od Androidu verze 1.5.

Druhým přístupem je psát celé tzv. *Activity* (spustitelná třída) v nativním kódu. K tomuto účelu existuje třída *NativeActivity*, nicméně tato možnost je dostupná až od Androidu verze 2.3. V implementační části této práce bude využívána prvně zmíněná možnost, proto zde bude detailněji rozebrána.

Java Native Interface

Na straně Java kódu je nejprve potřeba načíst požadované knihovny s nativními funkcemi. Toto se provádí až při samotném běhu aplikace z důvodu výběru správné verze knihovny zkompilované pro daný procesor (provedeno při instalaci aplikace). Poté je třeba deklarovat prototypy všech funkcí, které z načtených nativních knihoven budou v Java kódu volány. Klíčovým modifikátorem **native** je třeba určit, že se jedná o nativní funkce. Poté je možné funkce v Java kódu používat zcela běžným způsobem. Příklad načtení nativní knihovny a deklarace prototypu funkce (bez ošetření výjimek a chyb):

```
//Načtení knihovny nativeFunctions.so z adresáře libs ve složce projektu
static {
	System.loadLibrary("nativeFunctions");
}
//Prototyp funkce z dané knihovny
public native int nativeFunction (int[] array, int number, String name);
```

Na straně C/C++ kódu je situace o něco komplikovanější. Pro přístup k JNI rozhraní [15] je nutné importovat knihovnu *jni.h* ve zdrojových souborech, ve kterých bude k tomuto rozhraní přistupováno. Následně po zkompilování Java třídy, která obsahuje prototypy nativních funkcí je potřeba pomocí nástroje **javah** vygenerovat hlavičkový soubor v jazyce C, který bude obsahovat prototypy funkcí volaných z Java kódu dané třídy. Tímto se vytvoří propojení mezi Java kódem a kódem v C/C++.

Každá z takových funkcí je uvozena modifikátorem JNIEXPORT. Následuje návratový typ a klíčové slovo JNICALL. Název funkce je vygenerován podle odpovídajícího Java balíčku, názvu třídy a názvu samotné funkce. Jednotlivé části jsou odděleny znakem podtržítko. Následuje seznam parametrů funkce. Kromě parametrů deklarovaných v prototypu funkce na straně Java kódu, jsou automaticky při volání předávány další dva: ukazatel typu `JNIEnv*` umožňující interakci s JNI prostředím a referenci typu `jobject` na Javový objekt, který danou funkci zavolal.
Příklad (předpokládejme, že je funkce volána z třídy *MainClass* v balíčku *com.example.helloworld*):

```
//Prototyp funkce nativeFunction na straně kódu v jazyce C
JNIEXPORT jint JNICALL Java_com_example_helloworld_MainClass_nativeFunction
(JNIEnv *, jobject, jintArray, jint, jstring);
```

Při přístupu k parametrům [16] je třeba rozlišit, zdali se jedná o primitivní nebo objektové typy. K primitivním datovým typům lze přistupovat přímo, jsou totiž ekvivalentní s datovými typy jazyka C. Následující tabulka 1 obsahuje jejich seznam včetně velikostí.

Datový typ v Javě	Nativní datový typ	Velikost
boolean	jboolean	8 bitů, bez znaménka
byte	jbyte	8 bitů, se znaménkem
char	jchar	16 bitů, bez znaménka
short	jshort	16 bitů, se znaménkem
int	jint	32 bitů, se znaménkem
long	jlong	64 bitů, se znaménkem
float	jfloat	32 bitů
double	jdouble	64 bitů

Tabulka 1. Nativní typy a odpovídající Java datové typy [16].

Objektové datové typy nelze používat přímo, jsou totiž předávány jako ukazatele na vnitřní datové struktury virtuálního stroje. Nativní kód nemá informace o těchto strukturách. Navíc se zde objevují další komplikace. Například garbage collector by mohl daný objekt odstranit v době, kdy je k danému objektu přistupováno z nativního kódu. Proto je třeba k přístupu k objektovým typům využívat manipulační metody, které poskytuje JNI rozhraní. Tyto metody využívají parametr *JNIEnv**, který je automaticky předávám každé nativní funkci při jejím volání. Mezi objektové datové typy předávané odkazem patří také textové řetězce a pole. Zde je ukázka přístupu k řetězci znaků:

```
JNIEXPORT jint JNICALL Java_com_example_helloworld_MainClass_nativeFunction
(JNIEnv *env, jobject obj, jstring stringFromJava){

  const char *stringInC;
  stringInC = env->GetStringUTFChars(stringFromJava, NULL);

  if (stringInC == NULL) {
    //Chyba, nepodařilo se alokovat paměť
     return 1;
  }

  //Libovolná práce s řetězcem stringInC, např. výpis
  printf("%s", stringInC);

  //Uvolnění
  env->ReleaseStringUTFChars(stringFromJava, stringInC);
  return 0;
}
```

Pro přístup k textovým řetězcům poskytuje JNI rozhraní více funkcí, dále je zde ovšem zbytečné je rozebírat. Taktéž pro manipulaci s dalšími objektovými typy existují odpovídající metody, které je možné nalézt v dokumentaci JNI rozhraní [15].

Soubory Application.mk a Android.mk

Pro úspěšný překlad zdrojových souborů a sestavení výsledných nativních knihoven pomocí nástrojů NDK je nutné vytvořit odpovídající Android makefile soubory.

V rámci každého Android projektu s nativním kódem musí být vytvořen soubor **Application.mk** (informace o jeho obsahu byly získány z dokumentu APPLICATION-MK.txt, který je dostupný ve složce *docs* v archivu NDK). Jeho obvyklým umístěním je složka *jni* v projektové složce aplikace. Účelem tohoto souboru je popsat, které nativní moduly (knihovny) jsou aplikací požadovány. Tento soubor také definuje některé proměnné globálně ovlivňující překlad. Zde je příklad některých z nich:

- APP_PROJECT_PATH – absolutní cesta ke složce projektu.
- APP_MODULES – moduly požadované k sestavení.
- APP_CFLAGS – parametry předávané překladačům.
- APP_ABI – seznam instrukčních sad, pro které mají být nativní knihovny přeloženy.

Pokud není nastavena proměnná APP_ABI, jsou nativní knihovny překládány pro procesory s instrukční sadou *armeabi* (ARMv5TE). Dalšími možnostmi jsou sady *armeabi-v7a* nebo *x86*. Pokud

je uvedeno více typů instrukčních sad, dojde k překladu do všech verzí. Jednotlivé verze jsou po překladu uloženy ve složce *libs* v podsložkách odpovídajících dané instrukční sadě. K výběru odpovídající knihovny dojde až při instalaci aplikace na cílové zařízení.

V projektu dále musí být alespoň jeden soubor **Android.mk** (informace o jeho obsahu byly získány z dokumentu ANDROID-MK.txt, který je dostupný ve složce *docs* v archivu NDK). Opět se jedná o makefile soubor pro překladový systém NDK. Umožňuje seskupovat zdrojové C soubory do modulů, které pak tvoří sdílené nebo statické knihovny. Aplikace využívá pouze sdílené knihovny/moduly, pouze tyto jsou proto exportovány po skončení překladu do složky *libs*. Statické knihovny/moduly jsou využívány k vytvoření těch sdílených. Hlavičkové soubory zde není potřeba uvádět, ty jsou překladovým systémem zjištěny automaticky.

Příklad jednoduchého souboru Android.mk pro vytvoří sdílené knihovny *hello-jni.so* z jednoho zdrojového souboru *hello-jni.c*:

```
LOCAL_PATH := $(call my-dir)         //zjištění aktuální cesty

include $(CLEAR_VARS)                //zrušení globálně nastavených proměnných


LOCAL_MODULE    := hello-jni         //unikátní jméno modulu
LOCAL_SRC_FILES := hello-jni.c       //zdrojové soubory modulu

include $(BUILD_SHARED_LIBRARY)      //modul bude sdílenou knihovnou
```

Další příklady a podrobnosti lze samozřejmě dohledat na internetu. Nicméně velmi dobrým zdrojem informací jsou také dokumentační textové soubory v samotném balíčku NDK.

2.2.6 Struktura Android projektu

Všechny potřebné soubory k přeložení, sestavení a zkompletování výsledného *.apk* balíčku jsou obsaženy ve složce, která odpovídá dané aplikaci (projektu). Při vývoji aplikace pro Android je třeba zachovat určitou strukturu projektu, se kterou vývojové nástroje a překladače počítají.

AndroidManifest.xml

V kořenové složce je povinný soubor *AndroidManifest* [17]. Jedná se soubor formátu XML obsahující základní informace o aplikaci jako název balíčku (např. com.example.helloworld) a číslo verze Android API, kterou aplikace ke svému běhu potřebuje. Při instalaci balíčku na konkrétní Android zařízení poté dochází ke kontrole, zda-li cílový systém danou verzi API podporuje. Pokud ne, aplikace nebude nainstalována. Nové verze systému a spolu s nimi i nové verze Android API přichází relativně často, proto je právě tímto mechanismem zajištěno, aby nebyly aplikace využívající novější funkcionalitu instalovány na starší verze systémů.

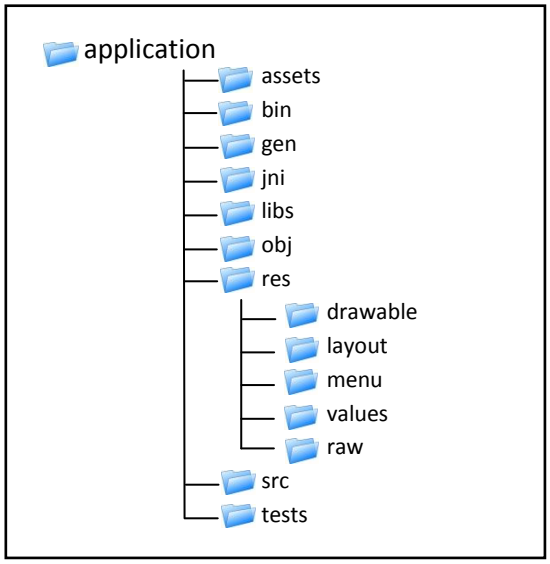
AndroidManifest dále obsahuje seznam všech Activity tříd (samostatné spustitelné třídy s uživatelským rozhraním) dané aplikace. Pokud by nějaká třída Activity nebyla uvedena a přesto se ji aplikace pokoušela spustit, ke spuštění nedojde. Dále může být u každé Activity zadán seznam tzv. Intents (jedná se o jakési abstraktní požadavky na operace, jde o formu komunikace a předávání dat mezi třídami typu Activity, případně také mezi službami běžícími na pozadí). Opět pokud chce daná třída Activity komunikovat pomocí Intents, musí zde být uvedeny.

Dalším prvkem souboru *AndroidManifest* je seznam oprávnění, která daná aplikace požaduje. Pokud například potřebuje aplikace číst nebo zapisovat data na externí paměťovou kartu, musí mít uveden požadavek na tento typ oprávnění (WRITE_EXTERNAL_STORAGE). Dalšími příklady oprávnění jsou ACCESS_FINE_LOCATION (přístup k GPS souřadnicím), CALL_PHONE (uskutečnit telefonní hovor), MODIFY_AUDIO_SETTINGS (měnit audio nastavení), a podobně. Tato oprávnění musí aplikace získat souhlasem uživatele během instalace. Při instalaci balíčku je tedy nejprve zobrazen seznam požadavků na oprávnění a ty musí uživatel potvrdit, teprve až potom může proběhnout instalace aplikace.

Soubor může obsahovat také další části, které jsou volitelné a nemusí být tedy vždy specifikovány. Zde patří například požadavek aplikace na obrazovku zařízení (velikost, hustota, počet), přítomnost určitého hardwarového zařízení (kamera, Bluetooth modul, hardwarová klávesnice, trackball…), požadavek na konkrétní verzi OpenGL ES, nebo i mnohé jiné požadavky, které jsou detailně popsány v oficiální dokumentaci.

Podsložky projektu

Android projekt je standardně členěn do dalších podsložek [18]. Jejich strukturu znázorňuje následující obrázek 9.



Obrázek 9. Struktura Android projektu.

Složka **assets** umožňuje do balíčku vložit další soubory, které může aplikace při běhu využívat. Na rozdíl od souborů ve složce **res** nejsou nezbytné, proto jim není automaticky generováno unikátní ID a při přístupu z aplikace je třeba testovat, zdali je daný soubor v balíčku skutečně obsažen.

Složka **bin** je vytvořena při kompilaci. Obsahuje podsložky se zkompilovanými Java třídami (*.class*), soubor *classes.dex* se spustitelným kódem aplikace pro Dalvik Virtual Machine a zkomprimovaný ZIP archiv *resources.ap_* se soubory ze složky **res.** Také je zde obsažen kompletní finální *.apk* balíček.

Složka **gen** obsahuje automaticky generovanou třídu R.java. Tato třída obsahuje dvojice vygenerovaných unikátních identifikátorů a názvů všech souborů umístěných ve složce **res**. Díky

tomu je možné z Java kódu aplikace jednoduše přistupovat k těmto souborům pomocí jejich ID resp. názvů.

Složka **jni** je přítomna pouze v případě, že aplikace využívá nativního kódu. Obsahuje zdrojové a hlavičkové soubory v jazyce C/C++. Dále jsou zde odpovídající makefile soubory *Android.mk* a *Application.mk* nutné k překladu pomocí NDK (viz. kapitola 2.2.4).

Do složky **libs** je možné umístit různé Java knihovny, které daná aplikace používá. V případě použití nativního kódu jsou zde po NDK překladu vytvořeny knihovny v nativním kódu (.so) ze zdrojových souborů obsažených ve složce **jni**. Tyto knihovny jsou navíc rozděleny do podsložek podle instrukční sady pro kterou byly přeloženy. Aplikace tedy může obsahovat více verzí nativních knihoven a podle cílového procesoru se při instalaci určí, která verze bude nainstalována.

Složka **obj** je vytvořena během překladu nativních zdrojových souborů pomocí NDK. Obsahuje jednotlivé zkompilované objektové soubory a další sobory potřebné pro kompilaci a linkování výsledné nativní knihovny.

Složka **res** obsahuje všechny nezbytné soubory, které aplikace využívá. Každému souboru je vygenerováno unikátní ID pomocí kterého lze k němu přistupovat z Java kódu aplikace. Soubory jsou rozděleny několika základních skupin (každá má odpovídající podsložku):

1. *drawable* – jedná se především o grafické soubory v podporovaných formátech. Tedy obrázky, ikony, pozadí a podobně. Mohou zde být různé verze obrázků (například pro různá rozlišení) rozděleny podle density obrazovky (low, mid, high). Podle schopností obrazovky cílového zařízení je automaticky použita odpovídající verze. V této skupině mohou být také XML soubory, které popisují vlastnosti některého z vložených grafických souborů.
2. *layout* – XML soubory, které obsahují popis rozložení prvků grafického uživatelského rozhraní. Umožňují měnit uživatelské rozhraní bez toho, aniž by musel být upravován zdrojový kód aplikace a prováděna kompilace. Také lze jednoduše vytvořit různé popisy rozložení grafických prvků pro různé velikosti a orientace obrazovek. Na cílovém zařízení je potom použit odpovídající layout soubor. Díky tomu lze velice snadno zajistit správné zobrazení aplikace na širokém spektru Android zařízení.
3. *menu* – XML soubory s popisem chování a obsahu různých menu nabídek.
4. *values* – XML soubory s dalšími hodnotami. Standardně je zde obsažen soubor *strings.xml* s textovými řetězci, převážně používanými v uživatelském rozhraní aplikace. Ten obsahuje seznam dvojic identifikátor-hodnota. V kódu aplikace se pak na místě použití řetězce zadává identifikátor, samotná textová hodnota je pak načtena ze souboru. Díky tomu je možné velmi snadno vytvářet různé jazykové verze aplikace, bez nutnosti zasahovat do kódu a provádět rekompilaci celé aplikace. Kromě textových řetězců lze vytvářet všemožné další seznamy.
5. *raw* – Další typy souborů, například audio nebo video.

V rámci složky **res** je možné vytvářet dále libovolnou strukturu podsložek a umísťovat externí soubory dle potřeb aplikace.

Složka **src** obsahuje zdrojové Java soubory. Ty jsou členěny do podsložek odpovídajících názvu daného balíčku, např. třída *Main* v balíčku *com.example.helloword* je uložena ve 3 vnořených složkách *com*, *example* a *helloworld*.

Složka **tests** obsahuje soubory automatického testování Java aplikací pomocí nástroje Junit. V projektu není povinná.

2.3 FFmpeg

FFmpeg [19] [20] je multimediální knihovna a balíček nástrojů pro nahrávání, konverzi, přehrávání a streamování audia a videa. Základem je knihovna kodeků *libavcodec* a knihovna *libavformat* pro práci s multimediálními formáty. FFmpeg je šířen zdarma pod licenci LGPL nebo GPL a má otevřené zdrojové kódy. Je taktéž multiplatformní. Vývoj probíhá na platformě Linux, nicméně použití je možné na celé škále operačních systému včetně Microsoft Windows, Apple Mac OS nebo Android. Taktéž je FFmpeg používán na mnoha typech procesorových architektur, jako x86, PowerPC, ARM nebo SPARC.

Zakladatelem projektu je *Fabrice Bellard*. Na vývoji a údržbě pracuje skupina *FFmpeg team*, která se převážně skládá z vývojářů a komunity okolo projektu *MPlayer*.

2.3.1 Základní součásti

- *ffmpeg* – nástroj běžící v příkazové řádce sloužící k převodu mezi multimediálními formáty a k real-time zachytávání videa z různých zdrojů (videokamera, TV tuner, webkamera).
- *ffserver* – streamovací server. Podporuje přenos video streamů přes protokol HHTP nebo RTSP. Podporuje taktéž time shifting (pozastavení a posun zpět v online streamech).
- *ffplay* – multimediální přehrávač využívající FFmpeg knihovnu. Využívá taktéž knihovnu SDL (Simple Directmedia Layer)
- *ffprobe* – nástroj zobrazující detailní informace o mediálních souborech. Funguje v příkazové řádce.
- *libavcodec* – základní knihovna pro kódování a dekódování audia a videa. Pro zajištění výkonnosti a optimalizaci byla většina kodeků implementována od základu znovu.
- *libavformat* – knihovna pro multiplexing a demultiplexing multimediálních kontejnerů.
- *libavutil* – knihovna pomocných algoritmů a funkcí (generátory náhodných čísel, datové struktury, matematické funkce, šifrovací/dešifrovací algoritmy a podobně).
- *libpostproc* – knihovna algoritmů pro video post-processing.
- *libswscale* – knihovna algoritmů pro změnu velikosti videa a konverzi barev.
- *libavfilter* – knihovna video filtrů a API pro jejich připojování.
- *libavdevice* – knihovna pro práci se vstupními a výstupními zařízeními pro zachytávání videa a rendering.

2.3.2 Knihovna libavcodec

Právě tato knihovna [21] z projektu FFmpeg bude využita v aplikaci pro samotné kódování a dekódování videa (krom dalších pomocných *libavformat, libavutil* a *libswscale*). Díky tomu, že je napsaná v jazyce C (norma C99), je možné tuto knihovnu přeložit a použít i na platformě Android. Je často využívána v open-source multimediálních přehrávačích jako *MPlayer, xine, KMPlayer* nebo *VLC*. Využívají ji také mnohé nástroje pro editaci videa jako *Kino, Cinelerra, CineFX* nebo *Kdenlive*. Vzhledem k podpoře mnoha audio kodeků je *libavcodec* používána také v mnoha přehrávačích a editorech zvuku jako *Audacity, SoX* nebo *RockBox*. Její použití a rozšíření je skutečně široké, například ji využívá i 3D editor *Blender*, multimediální centrum *MythTV*, webový prohlížeč *Google*

Chrome nebo knihovna počítačového vidění *OpenCV*. Známou aplikací na platformě Android používající tuto knihovnu je videopřehrávač *RockPlayer*.

Vzniklo také mnoho rozhraní pro použití v jiných jazycích jako *ffmpeg-php* pro jazyk PHP, *Jffmpeg* pro Javu a *Xuggler* pro jazyky Java a C++.

2.3.3 Podporované kodeky

Cílem vývojářů FFmpegu je přenositelnost, znovupoužitelnost kódu a výkonnost. Proto bylo potřeba většinu kodeků implementovat zcela znovu. Mnoho rozšířených kodeků je navíc proprietárních s nedostupnými zdrojovými kódy, proto byly mnohokrát využity metody reverzního inženýrství. Na rozdíl od originálních implementací jsou ty v *libavcodec* knihovně meziplatformě přenositelné a mnohdy i díky optimalizacím výkonnostně lepší. Nicméně daní za to mohou být občasné chyby, nepřesnosti, problémy s kompatibilitou určitých funkcí a podobně.

Přehled vybraných kodeků [21]

Audio	**Video**
• Sony: **ATRAC1, ATRAC3** • QuickTime audio: **ALAC, QDesign** • **MP1, MP2, MP3, AAC** • **Theora** • **Vorbis** • **Truespeech** • Windows Media Audio: **WMA1, WMA2, WMA Pro** • 3GP audio: **AMR-NB, AMR-WB** • DVD Forum: **Dolby, AC-3, TrueHD** • **GSM Full Rate** • RealPlayer: **RealAudio**	• vlastní kodeky: **FFV1, Snow** • **MPEG-1 Part 2** • **H.261, H.262/MPEG-2 Part 2, H.263** • **H.264/MPEG-4 AVC** • QuickTime video: **Sorenson 3, Cinepak, Motion JPEG** • Windows Media Video: **Microsoft Video 1, Cinepak, Indeo 2, 3, 5, Motion JPEG, Microsoft MPEG-4 v1, v2, v3, WMV1, WMV2, WMV3** • RealPlayer: **RealVideo** • Adobe Flash: **VP6, FLV, SWF**

Tabulka 2. Vybrané kodeky podporované knihovnou FFmpeg.

V rámci FFmpeg projektu vznikly také dva nové vlastní kodeky. Prvním z nich je *FFV1* (FF video codec 1). Jedná se o bezztrátový video kodek. Výhodou jsou zcela otevřené dostupné zdrojové kódy jak pro dekodér, tak pro kodér.

Druhým kodekem v rámci projektu je *Snow*. Nicméně stále ještě nebyla dokončena ani první kompletní verze, proto je kodek určen zatím pouze pro experimentální použití.

2.3.4 Podporované formáty

Taktéž podpora multimediálních formátů je bohatá. Zde je uvedena většina nejdůležitějších [19] [21]:

- **ASF** – Advanced Systems Format, proprietární formát firmy Microsoft určený především pro streaming audia a videa, je součást Windows Media frameworku.
- **AVI** – Audio Video Interleave, uvedeno Microsoftem. Jde o kontejner pro audio i video data, podporující více audio a video streamů.
- **FLV** – formát pro Adobe Flash Player, audio a video kontejner převážně pro přenos po internetu.
- **GXF** – General eXchange Format, určen pro přenos videoklipů ze vzdálených video serverů.
- **IFF** – Interchange File Format, multimediální kontejner uveden firmou Electronic Arts.
- **ISO media formáty** – QuickTime, 3GP a MP4.
- **Matroska** – otevřený multimediální kontejner. Umožňuje vložit do jednoho souboru neomezený počet audio a video stop, obrázků nebo titulků.
- **Maxis XA** – proprietární audio formát vyvinutý firmou Maxis.
- **MPEG program stream**
- **MPEG transport stream**
- **MXF** – Material eXchange Format. Audio a video kontejner pro profesionální použití. Definován standardy mezinárodní společnosti *Society of Motion Picture and Television Engineers*.
- **Ogg** – otevřený kontejner spravovaný sdružením *Xiph.Org Foundation*. Myšlenkou je svobodný formát nezatížený žádnými komerčními patenty.
- **OMA** – OpenMG audio. DRM systém společnosti Sony pro zabezpečení distribuce audio souborů ATRAC3, WAV nebo MP3.

3 Návrh a implementace aplikace

Tato kapitola se nejprve zabývá stručným návrhem aplikace a popisem její struktury se zaměřením na řešení klíčových problémů. Dále pak stručně popisuje postup implantace. Následně je popsáno uživatelské rozhraní a principy ovládání aplikace. Na konci kapitoly jsou uvedeny výsledky testů zaměřených na rychlost zpracování videa, jejich zhodnocení a v úplném závěru provedeno srovnání aplikaci s již existujícími.

3.1 Návrh aplikace

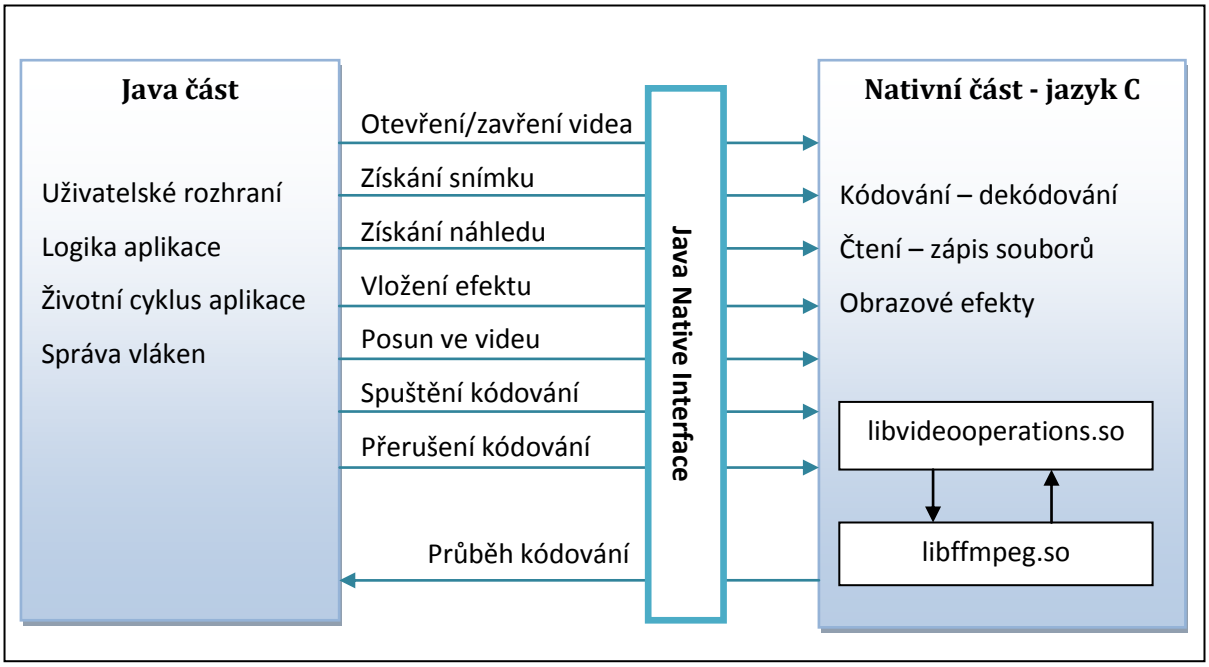
Aplikace bude rozdělena do dvou hlavních částí – Java a jazyk C. V Java části bude implementováno uživatelské rozhraní a hlavní funkční kostra aplikace. V jazyce C bude implementována nativní knihovna s výpočetně náročnými operacemi pracujících s videem. Využívat bude další nativní multimediální knihovnu FFmpeg.

Po spuštění aplikace bude moci uživatel otevírat video soubory různých formátů. Z nich půjde poté pomocí označení konkrétních časů vystřihovat nové klipy. Kterýkoliv z nich půjde poté vložit do časové osy a pomocí libovolných klipů tak poskládat nové výstupní video. Dále bude umožněno vkládat různé efekty na libovolná místa výstupního videa. Cílem bude implementovat práci s efekty tak, aby bylo v budoucnu možné a snadné do aplikace doimplementovat další nové efekty a rozšiřovat tak funkcionalitu aplikace. Po dokončení a zkompletování výsledku bude možné spustit kódování a vytvořit tak úplně nový video soubor přesně podle obsahu časové osy. Parametry výstupního formátu bude možné svobodně a libovolně nastavit dle potřeby.

Z pohledu uživatelského rozhraní bude snaha o jednoduchost ovládání a orientaci na použití dotykové obrazovky. Také bude zohledněna malá velikost displejů mobilních telefonů.

3.2 Struktura aplikace

Aplikace bude rozdělena do dvou částí, jedna bude implementována v jazyce Java a druhá v jazyce C (viz. obrázek 10). První část, jakási hlavní kostra, bude obsluhovat uživatelské rozhraní a řídit běh aplikace. V případě potřeby pak bude volat funkce z druhé části, která bude přiložena jako knihovna v nativním kódu. Zde budou implementovány veškeré operace s videem, jako otevírání, získávání snímků, dekódování, kódování, uzavření a podobně. Pouze v jednom případě je situace opačná, kdy je volána Java metoda z nativní části. Jedná se o pravidelné odesílání informací o průběhu při kódování výsledného videa. Komunikace mezi standardní Android Java aplikací běžící na virtuálním stroji a nativní částí probíhá pomocí JNI rozhraní (Java Native Interface).



Obrázek 10. Struktura aplikace.

3.2.1 První část – Java

Hlavní kostra programu je standardní Android aplikace, implementovaná v jazyce Java a spouštěna na virtuálním stroji. Složena je z několika tříd, zde je seznam těch nejpodstatnějších:

- **VideoEditor** – hlavní třída, která spouští aplikaci, zobrazuje a obsluhuje uživatelské rozhraní a volá v případě potřeby nativní metody.
- **EncodeSettings** – zobrazuje a obsluhuje okno s možnostmi nastavení parametrů výstupního videa (formát, rozlišení, bitrate, název souboru, kodeky a podobně).
- **EfeectSettings** – zobrazuje a obsluhuje okno s možnostmi nastavení parametrů obrazových efektů a jejich vytváření.
- **EffectInterface** – rozhraní, které musí implementovat každý obrazový efekt.
- **FileChooser** – třída s uživatelským rozhraním pro procházení souborového systému a předávání výsledku hlavní třídě. Slouží pro výběr souborů a jejich otevírání v editoru.
- **ClipLine** – třída reprezentující interaktivní panel s časovou osou. Umožňuje posun ve videu, označení počátečního a koncového střihu, zoomování a posunování ve videu.
- **TimeLine** – třída reprezentuje časovým měřítkem finálního videa. Jde o seznam videoklipů, ze kterých bude poskládáno finální video, včetně všech potřebných informací (délka, místa střihů apod.).
- **VideoClips** – uchovává seznam všech otevřených videoklipů a nově vzniklých výstřižků včetně všech jejich potřebných údajů.
- **VideoClipsEntry** – reprezentuje jednotlivý záznam seznamu VideoClips. Uchovává všechny potřebné informace o daném videoklipu, jako zdrojový video soubor, počátek, konec, aktuální pozici, časy střihů, bitmapu s náhledem a podobně.

Na straně Java kódu nejsou řešeny žádné zásadní operace aplikace, v podstatě se jedná hlavně o obsluhu prvků uživatelského rozhraní, naplnění datových struktur potřebnými daty a volání nativních funkcí. Proto zde není třeba tuto část dále rozebírat. Detailní informace o jednotlivých třídách (i těch zde neuvedených) lze samozřejmě v případě zájmu získat z komentářů přímo ve zdrojových souborech aplikace na přiloženém DVD. Za zmínku pak stojí ještě popis principu práce s obrazovými efekty.

Obrazové efekty

Aplikace poskytuje možnost aplikovat na výsledné video různé obrazové efekty. Pro demonstraci bylo implementováno několik ukázkových. Aplikace byla ale navržena tak, že není problémem doimplementovat další efekty a tímto rozšiřovat funkcionalitu aplikace. Bohužel je situace komplikována tím, že je část zdrojových kódů v jazyce C a část v jazyce Java. Samotné operace s jednotlivými hodnotami pixelů obrazu jsou z pochopitelných důvodů implementovány v nativní části aplikace. Tyto jsou proto popsány dále v kapitole 3.2.2 v části Obrazové efekty.

Na straně Java kódu je ovšem potřeba zajistit nastavení efektu a jeho vlastností (čas začátku, čas konce případně různé další parametry nastavující vlastnosti obrazové operace). Také je potřeba zajistit vizuální znázornění v časové ose videa. Proto je pro každý nově přidávaný efekt potřeba vytvořit nejen funkci v jazyce C provádějící samotnou operaci, ale i odpovídající Java třídu, která zajistí odpovídající akce uživatelského rozhraní a získání parametrů efektu.

Pro tento účel je přichystáno rozhraní *EffectInterface*. Každá třída obrazového efektu pak musí implementovat toto rozhraní. Každopádně díky tomuto mechanismu pak stačí jen implementovat potřebné metody tohoto rozhraní v nově vytvářené efektové třídě a přidat na odpovídající místo kódu vytvoření její instance. Relativně jednoduše tak lze rozšiřovat nabídku dostupných efektů. Detailní praktické pokyny vytvoření nového efektu jsou pak uvedeny v komentářích zdrojového souboru třídy *EffectSettings*.

3.2.2 Druhá část – jazyk C

Veškeré operace s videem jsou implementovány v jazyce C a přeloženy do souborů nativních knihoven *libffmpeg.so* a *libvideoOperations.so*. Knihovna *ffmpeg* pak obsahuje kompletní přeloženou knihovnu FFmpeg jejíž funkce jsou volány z knihovny *videoOperations*. Zdrojové soubory knihovny FFmpeg jsou k diposzici pod licencí GPL (General Public License) a jsou k dispozici na webových stránkách projektu. Nicméně pro úspěšný překlad pomocí Android NDK je potřeba použít upravenou verzi, která je k dispozici z různých internetových zdrojů.

Knihovna *videoOperations* osahuje veškeré funkce, které jsou volány z Java části aplikace pomocí rozhraní JNI a také funkce a struktury, které jsou dále potřeba. Skládá se ze dvou zdrojových souborů *videoOperations.h* a *videoOperations.c*. První z nich, hlavičkový soubor, je automaticky vygenerovaný pomocí nástroje javah (viz. kapitola 2.2.5 sekce Java Native Interface). Tím je zajištěno korektní vygenerování prototypů funkcí a jejich správné napojení na JNI rozhraní. Druhý soubor obsahuje implementaci všech potřebných datových struktur a funkcí.

Následující sekce popisují nejdůležitější operace v nativní části a zjednodušený princip jejich fungování.

Otevření videa

Otevření nového video souboru zajišťuje funkce `openVideo` kterou lze zavolat pomocí JNI rozhraní z Java kódu. Protože se jedná i o první volanou nativní funkci, zajistí registraci a uvedení do činnosti

všech potřebných součástí FFmpeg knihovny a také provede alokaci potřebných datových struktur. Protože je funkce volána při otevírání každého videosouboru, je toto provedeno pouze při prvním volání funkce.

Jedním z parametrů je cesta k souboru, který má být otevřen. To je provedeno pomocí FFmpeg funkce `av_open_input_file`. Následuje vyhledání video a audio streamů. Pokud není nalezen žádný video stream, soubor je odmítnut. Naopak audio stream nemusí být v souboru přítomen žádný.

Pro nalezené streamy jsou vyhledány odpovídající dekodéry (`avcodec_find_decoder`). Pokud nejsou požadované dekodéry v knihovně FFmpeg dostupné, opět nelze se souborem v případě absence video dekodéru dále pracovat. V případě, že jsou dekodéry k dispozici, je provedena jejich inicializace (`avcodec_open`). Když inicializace u video dekodéru selže, opět nelze se souborem dále pracovat.

Uzavření videa

Uzavření video souboru zajistí funkce `closeVideo`, kterou lze volat pomocí JNI. Zajistí korektní uzavření konkrétního otevřeného videosouboru. To znamená jednak korektní uzavření video dekodéru a uvolnění jeho zdrojů (`avcodec_close`) a také uzavření na úrovní vstupního souboru (`av_close_input_file`).

Posun ve videu

Velmi často je potřeba přesunout se na určitou pozici ve videu, to zajistí funkce `seekVideo`. Té je předána hodnota požadovaného času a číslo videa, ve kterém je posun požadován. Byla zvolena reprezentace, kterou FFmpeg často využívá, konkrétně hodnota 1 000 000 odpovídá reálně jedné sekundě. Funkce `seekVideo` je volána jak pomocí JNI z Java části aplikace, tak z jiných funkcí v rámci nativní knihovny.

K rychlému posunu v multimediálním souboru je využita knihovní FFmpeg funkce `av_seek_frame`. Umožňuje posun oběma směry na libovolný nejbližší snímek, resp. datový paket k zadanému času. V našem případě je ale využit posun na nejbližší klíčový snímek před požadovaným okamžikem, důvody budou vysvětleny později.

Získání ikony videa

Získání náhledu z požadovaného videa zajistí funkce `getThumbnail`. Tato funkce je dostupná přes rozhraní JNI. Jedním z parametrů je odkaz na pole, které obsahuje bitmapu zobrazovanou Java uživatelským rozhraním aplikace.

Po alokaci všech potřebných zdrojů je přečten nejbližší další datový paket ze zadaného video souboru (`av_read_frame`). Pokud se nejedná o paket z video streamu, jsou čteny další pakety, než nastane úspěch. Video paket je následně dekódován do YUV reprezentace (`avcodec_decode_video`). Nicméně bitmapa pro zobrazení v uživatelském rozhraní aplikace musí být ve formátu RGB, proto je potřeba provést konverzi. Navíc má bitmapa náhledu určitou velikost v závislosti na rozlišení displeje, proto je také nutné převzorkovat rozlišení. Toto umožní po správném nastavení a alokaci potřebných struktur FFmpeg funkce `sws_scale` (viz. obrázek 11).

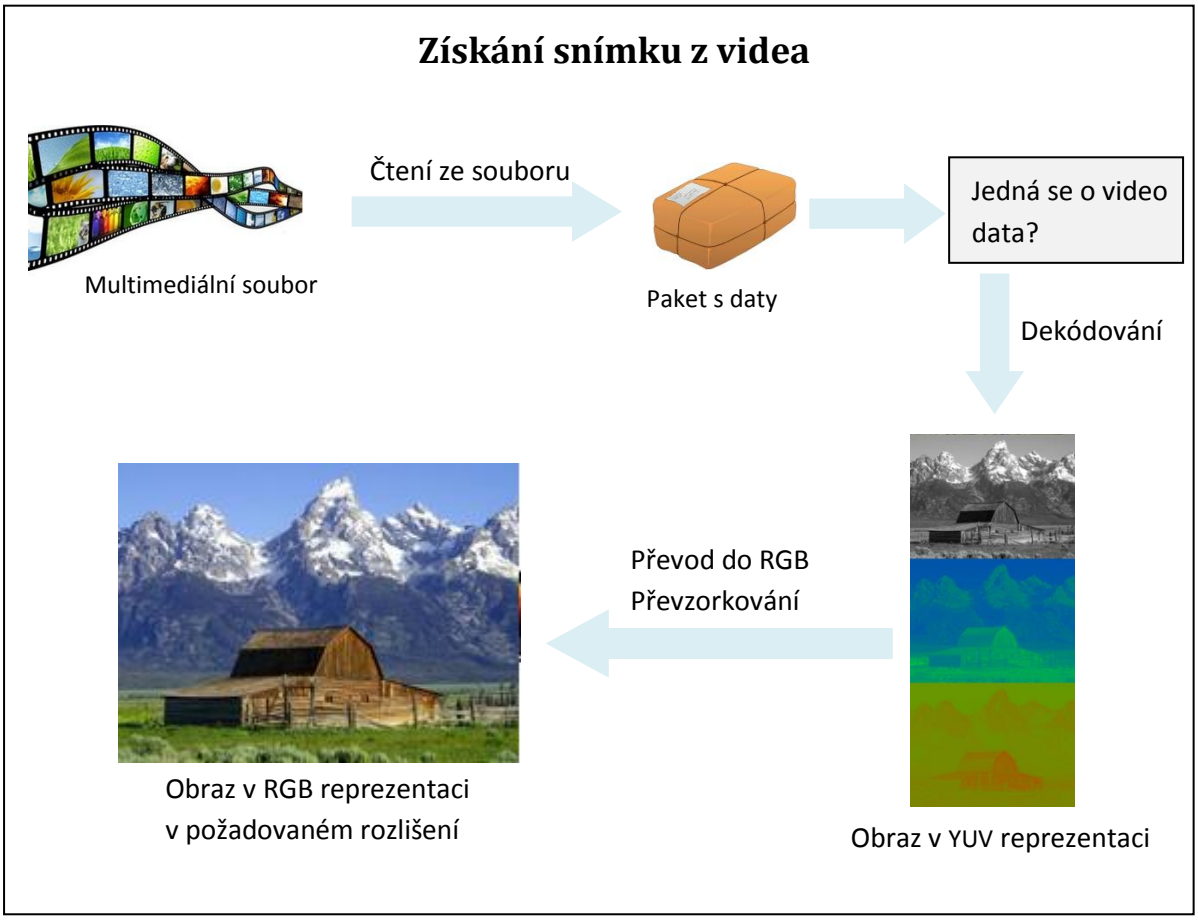
U video souborů se lze často setkat s tím, že na začátku je použito postupné roztmívání obrazu, proto často bývá první snímek videa celý černý. Toto není zrovna ideální snímek pro náhled videa, proto je před voláním funkce `getThumbnail` proveden posun o několik sekund vpřed od začátku videa. Posun je proveden na klíčový snímek, proto je náhled vždy kompletní a bez artefaktů (ty by

byly viditelné po přesunu a dekódování snímku, který by klíčový nebyl). Po získání náhledu je opět video posunuto do původní pozice, tj. na začátek.

Získání snímku z videa

Přečtení následujícího snímku ve zvoleném videu zajistí funkce `getFrame`. Je volána z Java kódu, poté, co uživatel vybere v panelu s časovou osou nový časový okamžik. Funkce je také volána při přehrávání videa. Před jejím zavoláním je proveden posun na požadovaný čas pomocí funkce `seekVideo`. To neplatí v případě přehrávání videa. Ta posune pozici ve videu ovšem pouze na nejbližší předchozí klíčový snímek k danému okamžiku. Proto je potřeba číst více datových paketů, dekódovat je a kontrolovat jejich časový údaj, který udává čas zobrazení snímku daného paketu na výstupu. Čtení paketů končí, když je přečten paket s nejbližším časovým údajem. Obvykle nemůže být vybrán paket s úplně přesně shodným časovým údajem, protože uživatelským rozhraním může být vybrán časový okamžik, který přesně neodpovídá času umístění jednotlivých snímků. Uživatelské rozhraní toto nijak nehlídá, není to totiž ani potřeba.

Získaný snímek je dále dekódován do formátu YUV. Pro zobrazení v uživatelském rozhraní je potřeba konverze do RGB barevného prostoru a také je třeba změnit rozlišení. To je provedeno podobně jako ve funkci `getThumbnail`. Princip získání snímku z videa je znázorněn na následujícím obrázku 11.



Obrázek 11. Princip získání jednoho snímku z videa.

Kódování videa – hlavní cyklus

Kódování výsledného videa je stěžejní a nejvíce rozsáhlou částí nativní knihovny *videoOperations*. Zajišťuje poskládání výstupního videa, aplikaci efektů a zakódování do výstupního formátu. Není implementováno v jedné velké funkci, ale je složeno z mnoha vzájemně se volajících funkcí.

Vstupním bodem je funkce `startEncoding`, která je z Java části volána pomocí JNI rozhraní. Funkci jsou předány požadované parametry výstupního formátu a informace o klipech, které tvoří výsledné video. Jedná se o jejich pořadí, začáteční časy, koncové časy a čísla zdrojových video souborů (přesně tak jak jsou poskládány v časové ose). Dalším krokem je inicializace všech potřebných proměnných, front, bufferů a datových struktur. Následuje otevření souboru pro zápis a nastavení výstupního formátu. Podle požadovaných výstupních parametrů jsou přidány a nastaveny výstupní video a audio streamy. Následuje inicializace a otevření odpovídajících video a audio kodeků. Pokud vše proběhlo hladce, je do výstupního souboru zapsána hlavička formátu. Následuje hlavní cyklus, který zavolá zpracování dalšího přečteného datového paketu ze vstupního videa (viz. následující část Kódování videa – zpracování paketu), jeho zakódování a posun času ve výstupních streamech. Tento cyklus je ukončen, pokud je výsledné video celé zpracováno (tj. celé časová osa dokončena), nebo dokud uživatel zpracování nezastaví. Na konec je formát korektně ukončen, uzavřen zápis do souboru a uvolněny alokované prostředky.

Kódování videa – zpracování paketu

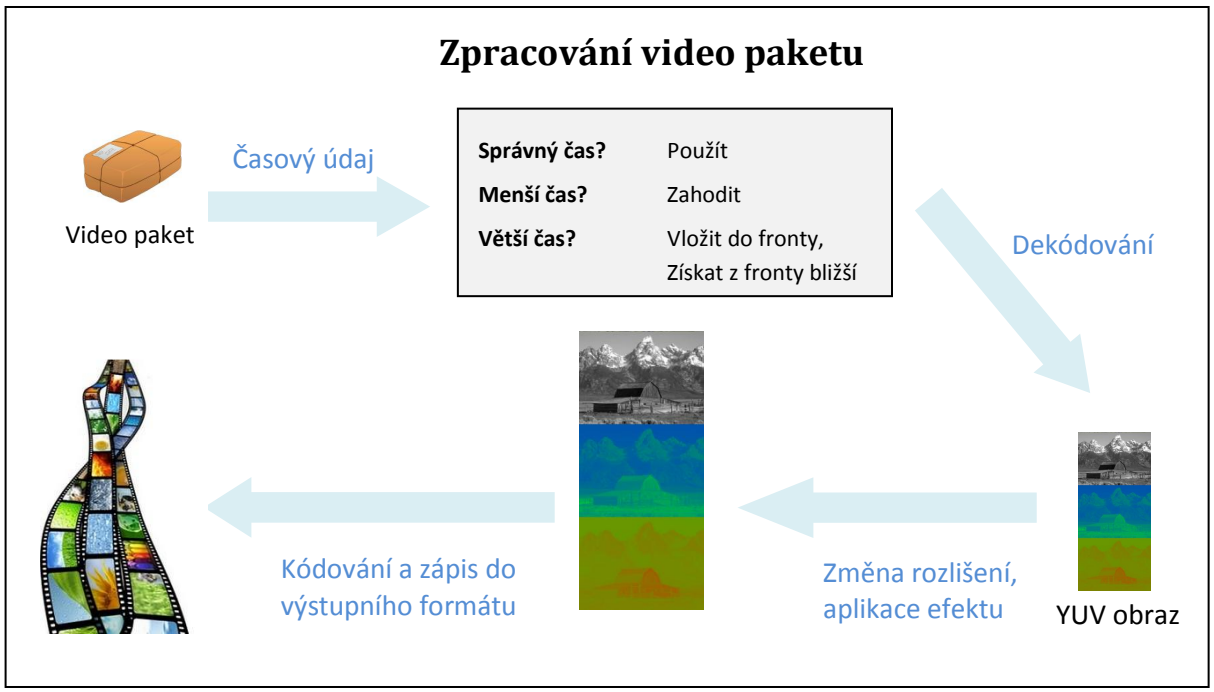
V předchozí části byl popsán hlavní cyklus při kódování výstupního videa. Pro větší přehlednost bude stěžejní část, kterou je *zpracování každého přečteného vstupního paketu*, popsána v této samostatné části textu.

Při zavolání funkce, která zpracování provádí, je znám aktuální čas ve výstupním videu. Proto je možné určit, ze kterého zdrojového videa je potřeba číst další datový paket. V čase 0 se tedy začíná s prvním klipem v časové ose v čase jeho začátku. Další pakety jsou potom čteny právě z video souboru odpovídajícímu tomuto klipu. Poté, co čas výstupního videa překročí koncový čas prvního zdrojového klipu, je druhý klip z časové osy posunut na svůj začáteční čas a další čtení paketů probíhá ze zdrojového videa právě tohoto druhého klipu. To pokračuje opět do okamžiku jeho koncového času. Takto jsou zpracovány všechny klipy v časové ose až do dokončení posledního klipu. Posuny na počáteční časy jednotlivých klipů jsou prováděny na paket s časem nejbližším k požadovanému času.Po přečtení datového paketu je potřeba odlišit, zdali se jedná o paket z video nebo audio streamu. Každý musí být pochopitelně dále zpracován jiným způsobem.

Nejprve se podívejme na zpracování **video paketu**. Počet snímků za sekundu ve zdrojovém i cílovém videu může být odlišný. Proto je potřeba toto zohlednit. Každý datový paket obsahuje informaci o čase, kdy by měl být prezentován na výstupu. Pokud je zjištěno, že čas paketu leží v rozsahu aktuálního výstupního snímku, je možné ho ihned použít. Pokud je zjištěno, že leží časově v předstihu mimo čas aktuálního výstupního snímku, může být zahozen a čten další paket (to je případ, kdy je počet snímků za sekundu ve vstupním videu větší než ve výstupním). Pokud nastane opačná situace, kdy paket leží až za koncovým časem aktuálního výstupního snímku, je potřeba tento paket vložit do fronty pro budoucí zpracování a v aktuálním čase použít už dříve zpracovaný paket (resp. jeho snímek). Ten je nejprve hledán průchodem fronty od začátku. Zde opět mohou nastat různé případy. Pokud je nalezen paket s vyhovujícím časem, je použit. Pokud není ve frontě nalezen (prázdná fronta, všechny pakety mají vyšší čas než je potřeba, všechny pakety mají menší čas než je potřeba), tak je použit poslední úspěšně dekódovaný snímek. Ten je v tomto případě tím časově nejbližším. Pokud je při práci s frontou zjištěno, že paket je dále nepotřebný, je odstraněn. Situace

s použitím fronty nastává, pokud je počet snímků za sekundu vstupního videa menší než počet snímků výstupního.

V této fázi je k dispozici paket nejlépe odpovídající času aktuálního výstupního snímku. Nyní může být paket dekódován (`avcodec_decode_video`). Získáme obraz snímku v YUV reprezentaci. Pokud se rozlišení liší od rozlišení výstupního videa, je potřeba obraz odpovídajícím způsobem převzorkovat. Následně je potřeba otestovat, jestli v daný čas není požadována aplikace obrazového efektu případně více efektů. Pokud ano, je toto provedeno (viz. kapitola 3.2.2 část Obrazové efekty). Zpracování snímku je dokončeno jeho zakódováním výstupním video kodérem (`avcodec_encode_video`) a zapsáním do výstupního souboru. Následuje posun času výstupního video streamu. Pro názornost je princip znázorněn na obrázku 12.



Obrázek 12. Zpracování video paketu během kódování výsledného videa.

Pokud je přečten **zvukový paket** v odpovídajícím čase, je dekódován (`avcodec_decode_audio2`). V případě, že se liší vzorkovací frekvence nebo počet kanálů oproti výstupním audio parametrům, je potřeba provést odpovídajíc převzorkování. To je provedeno pomocí `audio_resample` knihovní funkce, která je před použitím správným způsobem nastavena. Výsledek převzorkovaných dat je ukládán do FIFO bufferu. Každý audio kodek totiž vyžaduje při kódování výstupního audio paketu jinou velikost vstupních dat. Požadovaný počet bytů si tak potom čte z fronty, která je průběžně naplňována přečtenými a požadovaným způsobem převzorkovanými vstupními zvukovými vzorky. Po zakódování jsou nové audio pakety zapisovány do výstupního formátu. Princip je znázorněn na obrázku 13.

Obrázek 13. Zpracování audio paketu během kódování výsledného videa.

Obrazové efekty

Jak bylo uvedeno dříve (viz. 3.2.1 část Obrazové efekty), implementace samotných algoritmů pro úpravu obrazu jsou umístěny v nativní části aplikace. Zatímco v Java části je přichystáno rozhraní pro třídy nových efektů (respektive interakce s uživatelským prostředím, nikoliv samotná práce s obrazem), ve zdrojových souborech v jazyce C je situace pochopitelně odlišná.

Každý obrazový efekt je implementován ve svojí vlastní funkci. Ta je volána po získání snímku ze zdrojového videa před jeho zakódováním výstupním kodérem, pokud je v daném čase snímku obrazový efekt požadován. Toto je zjištěno tak, že pro aktuální čas ve výstupním videu je prohledán seznam všech použitých efektů, který mimo jiné obsahuje pro každý efekt počáteční a koncový čas. Po dekódování snímku ze zdroje je získán obraz v YUV reprezentaci. Obvykle algoritmy upravující obraz ovšem pracují s hodnotami RGB, proto je před voláním funkce efektu obraz do RGB převeden. Po úpravě efektem je převeden zpět do YUV, protože ten je požadovaným formátem barev pro FFmpeg video kodéry. Samotná konverze barevného formátu je prováděna pomocí rychlých a optimalizovaných funkcí FFmpeg knihovny.

Příklad prototypu obecné funkce efektu:

```
int exampleEffect(AVFrame *inputFrame,int width, int height,
int filterNumber, uint64_t actualTime);
```

Z příkladu lze vidět, že funkce efektu má k dispozici ukazatel na strukturu, která obsahuje barevné hodnoty jednotlivých pixelů, dále pak rozměry obrazu, číslo v seznamu efektů a nakonec čas aktuálního snímku. Číslo v seznamu efektů umožňuje identifikovat konkrétní efekt a jeho další parametry. Kromě počátečního a koncového času efektu (pro samotnou funkci obrazové úpravy většinou nepodstatné údaje) umožňuje získat pole s dalšími libovolnými parametry, které ke svému

správnému provedení potřebuje. Parametr s aktuálním časem je přítomen z důvodu, aby bylo možno implementovat také různé časově závislé efekty.

Tento mechanismus umožňuje relativně jednoduše rozšířit aplikaci o další efekty. Popis uvedený zde byl podán velmi stručně, detailní informace a příklady lze nalézt v komentářích ve zdrojových kódech aplikace. Několik efektů bylo pro ukázku implementováno, jejich popis je uveden v následujících odstavcích.

1) Šedotónový obraz

Tento jednoduchý filtr převede barevný obraz do šedotónového [22]. Nepoužívá žádné další parametry ani aktuální čas snímku. Z jednotlivých barevných RGB složek v každém pixelu je vypočítána intenzita. Tato hodnota je poté nastavena stejně do všech tří barevných kanálů výsledného obrazu. Princip výpočtu je založen na vnímání jednotlivých barevných složek lidským okem. Použita byla rovnice 1 pro výpočet intenzity, kde *I* je výsledná intenzita, *R* je červená složka barvy, *G* zelená a *B* modrá.

$$I = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \qquad (1)$$

2) Sépiový filtr

Tento filtr [23] je často používán, převádí obraz do hnědých tónů a navozuje efekt starobylosti. Nepoužívá žádné další parametry ani aktuální čas snímku. Upravuje hodnoty všech tří barevných RGB složek. Existuje velké množství variant lišících se v použitých konstantách, v implementaci aplikace byly zvoleny následující (*rovnice 2*), kde *R* je původní červená složka, *G* zelená a *B* modrá.

$$\begin{aligned} newR &= 0{,}393 \cdot R + 0{,}769 \cdot B + 0{,}189 \cdot G \\ newG &= 0{,}349 \cdot R + 0{,}686 \cdot B + 0{,}168 \cdot G \\ newB &= 0{,}272 \cdot R + 0{,}534 \cdot B + 0{,}131 \cdot G \end{aligned} \qquad (2)$$

U takto zvolené varianty konstant je navíc třeba hlídat překročení maximální hodnoty 255 pro každou novou barevnou složku.

3) Prahování

Ukázka efektu [22], který potřebuje další přidané parametry – konkrétně požadovanou hodnotu prahu. U každého pixelu je spočítána hodnota intenzity jako u šedotónového efektu a výsledek je porovnán s hodnotou prahu (0-255). Pokud leží v intevalu 0 až hodnota prahu, výsledkem je 0 ve všech barevných složkách. Pokud leží intenzita v intervalu hodnota prahu až 255, výsledkem je 255 ve všech barevných složkách. Výsledkem je tedy černobílý obraz.
Výpočet popisují následující *rovnice 3*, kde *x* a *y* jsou souřadnice pixelu, *I(x,y)* je funkce pro výpočet intenzity – viz. rovnice 1, *O(x,y)* je výstupní obraz a *T* hodnota prahu.

$$\begin{aligned} O(x,y) &= 0 \quad pro\ I(x,y) < T \\ O(x,y) &= 255 \quad pro\ I(x,y) \geq T \end{aligned} \qquad (3)$$

3.3 Postup implementace

Nejprve bylo potřeba seznámit se s platformou Android a vývojem aplikací, včetně odzkoušení rozmanitých ukázkových tutoriálů a zdrojových kódů. K tomu bylo potřeba získat veškeré vývojové nástroje, nainstalovat je a zprovoznit. Následně bylo nutné seznámit se s principem vývoje nativních aplikací na této platformě a fungováním překladového balíčku NDK.

Dále bylo potřeba nastudovat možnosti knihovny FFmpeg, seznámit se základními funkcemi a principy a zvážit, zdali bude tato knihovna skutečně vhodná po požadovaný účel. Dále bylo potřeba získat zdrojové soubory knihovny FFmpeg a zjistit, jak provést úspěšný překlad pomocí NDK do nativního kódu pro Android. Tato fáze byla nečekaně časově náročná, protože bohužel nelze jednoduše použít zdrojové soubory dostupné na webových stránkách projektu. Je totiž potřeba zdrojové soubory patřičně upravit. Naštěstí se ale podařilo nalézt upravenou verzi vhodnou pro Android. Nicméně pro úspěšný překlad pomocí NDK bylo potřeba dále upravit makefile soubory *Application.mk* a *Android.mk* (viz. kapitola 2.2.5). Jisté návody nebo tutoriály existují, bohužel ne všechny vedou k úspěchu. Verzí NDK totiž existuje více, navíc došlo od jisté verze k radikální změně překladu. Také verzí FFmpeg knihovny existuje více, včetně různě upravených zdrojových souborů pro Android.

Samotný překlad je časově náročný, na běžném desktop PC trvá přibližně 20 minut. Překlad knihovny FFmpeg bylo třeba provést naštestí pouze několikrát, veškeré další úpravy a změny kódu probíhaly ve zdrojových a hlavičkových souborech knihovny *videoOperations*. Její rozsah je daleko menší, a proto překlad trval obvykle v řádu sekund.

V prvních fázích implementace bylo potřeba vytvořit kostru aplikace a jednoduché uživatelské rozhraní, které umožňovalo základní interakci s video soubory, tzn. otevření, získávání snímků a korektní uzavření. Toto bylo dále vylepšeno o třídu pro průchod souborovým systémem a galerii video klipů s náhledy z jednotlivých video souborů.

Následovala implementace interaktivního panelu s časovou osou, který umožňuje vybírat místo náhledu z videa a nastavovat počáteční a koncový čas střihu. Po vytvoření tohoto panelu bylo možné implementovat funkcionalitu umožňující vystřihávat z videoklipů zvolené části a vytvářet z nich další samostatné klipy. Nyní bylo možné vytvořit finální časovou osu, do které je možné jednotlivé klipy vkládat, měnit jejich pořadí nebo je z osy odstraňovat.

Dalším krokem byla implementace sestavení výsledného videa z jednotlivých klipů ve finální časové ose v zadaném pořadí a jeho samotného kódování do zvoleného výstupního formátu. Práce na implementaci kódování výsledného videa byla bezesporu nejnáročnější částí. Informací a praktických příkladů (obzvláště pak těch funkčních) na kódování pomocí FFmpeg knihovny je velice málo. Obecně se informace o FFmpeg API získávají značně složitě. Z těch oficiálních existuje zastaralý tutoriál na principy dekódování, dále velmi stručně komentovaná Doxygen dokumentace vygenerovaná ze zdrojových souborů a dva příklady (obecná práce s API a kódování MPEG videa s jednoduchým vygenerovaným obrazem) ve zdrojovém archivu projektu. Další informace lze pak získat především na různých vývojářských fórech. Proniknutí do práce s FFmpeg knihovnou je tak pro neznalého člověka relativně časově náročné, nehledě na to, že v Android verzích FFmpegu existují určité odlišnosti.

Jedním z posledních kroků byla implementace obrazových efektů a mechanismu jejich začlenění do aplikace. V závěrečných fázích implementace probíhalo intenzivní testování a hledání chyb, optimalizace kódu a práce na vylepšení uživatelského rozhraní a vzhledu celé aplikace. Při testování výsledných videí se pak osvědčilo použití přehrávače *VLC*, který je postaven také na FFmpeg knihovně a proto podporuje stejné formáty a kodeky.

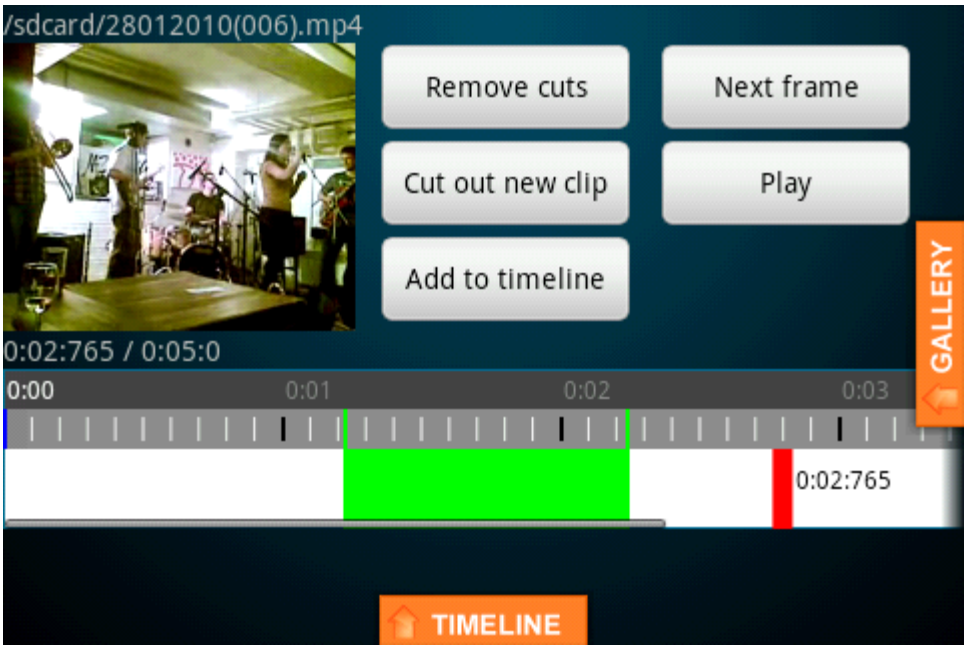
3.4 Uživatelské rozhraní

Vzhledem k charakteru aplikace je uživatelské rozhraní orientováno pevně na šířku, bez možnosti přepnutí do vertikální orientace. Ovládací prvky jsou uzpůsobeny dotykovému ovládání. Hlavní editační stránka, pokud je otevřeno nějaké video, je rozčleněna do několika základních sekcí.
V levé části je náhled právě zvoleného snímku z videa. Je zobrazen snímek odpovídající času, který je aktuálně zvolen v dolním ovládacím panelu. Čas nemusí odpovídat zcela přesně na milisekundy, je použit nejbližší snímek ve videu. Aktuální čas ve videu a celkový čas klipu v textové podobě jsou zobrazeny pod náhledem. Vedle náhledu vpravo je skupina ovládacích tlačítek:

- **Tlačítko pro střih** – v aktuálně zvoleném čase je vytvořen střih. Po vytvoření dvou střihů je oblast mezi nimi označena a je možné tento úsek videa vystřihnout do nového klipu nebo střihy zrušit.
- **Vystřižení nového klipu** – pokud jsou označena dvě místa střihu, tato volba vytvoří nový klip definovaný časy střihů. Při vytváření lze také upravit pojmenování klipu pro snazší identifikaci. Nový klip je pouze virtuální, žádný nový video soubor není vytvořen. Do galerie je přidána ikona nového klipu a lze s ním následně pracovat jako s kterýmkoliv jiným běžným klipem.
- **Vložení do časové osy** – aktuální klip je vložen do časové osy. Ta reprezentuje výsledné výstupní video. Do časové osy lze vkládat jak standardní videa otevřené ze souborového systému, tak i klipy vystřižené z jiných videí.
- **Přehrávání** – spustí přehrávaní aktuálního videa. Nejedná se ovšem o reálnou rychlost a zvuk se nepřehrává. Slouží pouze pro snadnější orientaci ve videu. Pokud je přehráváno video ve velkém rozlišení, může být přehrávání značně pomalé (dáno hardwarovými možnostmi konkrétního zařízení).
- **Další snímek** – posune video o jeden snímek dopředu. Slouží pro přesný posun po jednotlivých snímcích. Posun na předchozí snímek není implementován, tato funkce není na straně FFmpegu podporována a řešení by bylo značně složité a pomalé.

Ve spodní části obrazovky je dotykový panel s časovým měřítkem a červeným kurzorem ukazujícím aktuální pozici ve videu. Umožňuje uživateli posunovat se na požadovaný čas ve videu. Je třeba vzít na vědomí, že posun ve videích s velkým rozlišením může být časově náročný a není okamžitý. Pokud zařízení podporuje multitouch, je možné pomocí dvou prstů časovou osu zoomovat. Pokud zařízení multitouch nepodporuje, je zobrazen navíc posuvník, který zoomování umožní.

Z boční a spodní strany obrazovky je možné vysunout galerii klipů a časovou osu výsledného videa. Galerie umožňuje otevírat nové video klipy ze souborového systému, zobrazovat informace o videoklipech a vybírat, se kterým klipem bude aktuálně pracováno. Vysunovací panel s výslednou časovou osou zobrazuje sekvenci klipů, které tvoří výsledné video, umožňuje vkládat obrazové efekty a spouštět kódování výsledného videa.

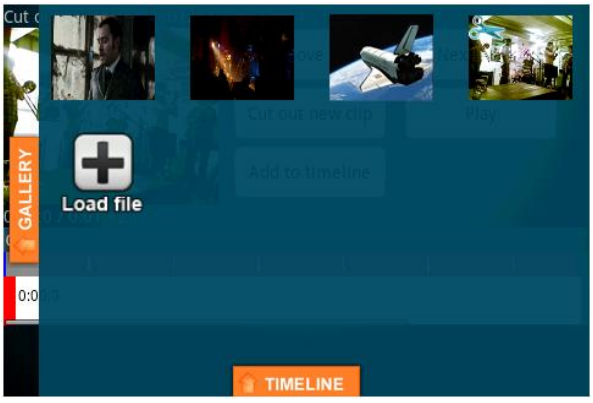


Obrázek 14. Hlavní obrazovka aplikace.

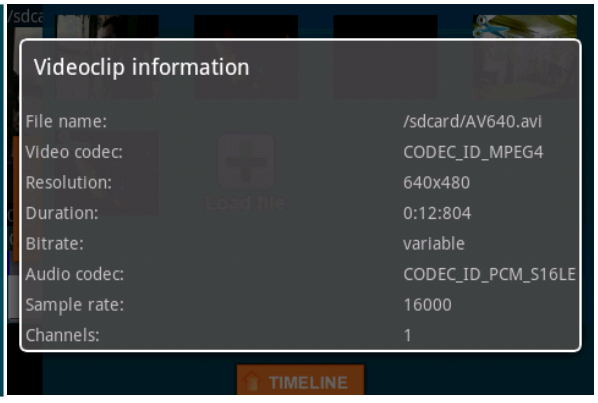
3.4.1 Galerie videoklipů

Z pravé strany je možné vysunout panel s galerií videoklipů. Kliknutím na ikonu „Load file“ je spuštěna třída umožňující průchod souborovým systémem a výběr video souboru. Po vybrání je souboru otevřen (pokud je podporován) a v galerii je zobrazena ikona s náhledem. Pokud galerie obsahuje více klipů, než se vleze na plochu displeje, je možné v galerii scrollovat. Pokud je vytvořen nový videoklip vystřižením z již existujícího klipu, jeho náhled se taktéž objeví v galerii. Odlišen je pomocí ikony nůžek v levém horním rohu náhledu.

Galerie slouží také k výběru klipu, se kterým se bude pracovat v hlavní sekci. K tomu stačí pouze dotek na příslušný náhled a tím je videoklip zvolen. Dlouhým dotekem jsou zobrazeny detailní informace o daném videu.



Obrázek 16. Galerie videoklipů.



Obrázek 15. Informace o videu.

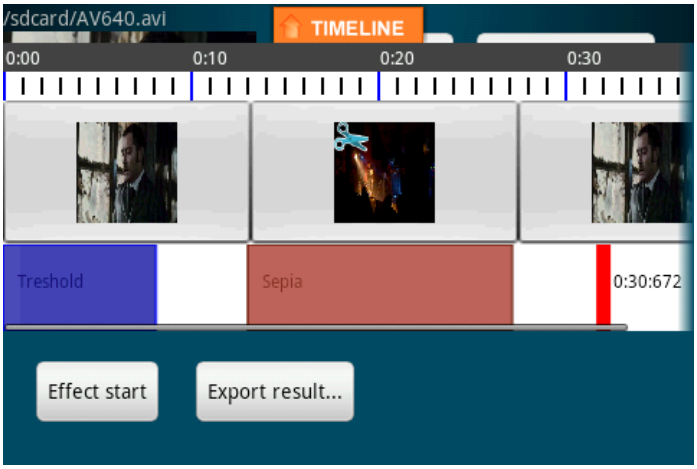
3.4.2 Časová osa

Vysunovací panel s časovou osou reprezentuje výsledné video. V horní části zobrazuje časové měřítko, uprostřed vložené klipy s náhledy a dole oblast pro vkládání obrazových efektů. Pokud zařízení podporuje multitouch, je možné časovou osu zoomovat pomocí dvou prstů (v oblasti časového měřítka nebo efektů). V opačném případě je ve spodní části zobrazen navíc posuvník umožňující zoomování provést.

Pokud je potřeba klip z výsledku odebrat, přesunout na jinou pozici nebo zkopírovat, umožní to kontextové menu zobrazené po dlouhém doteku na konkrétní klip. V případě přesunování nebo kopírování klipu je poté potřeba dlouhým dotekem opět do oblasti klipů určit nové místo vložení.

V oblasti klipů je možné vybrat pomocí červeného kurzoru konkrétní čas ve výstupním videu. Pomocí tlačítka vlevo dole lze označit vybrané místo jako počáteční čas efektu. Po označení koncového času efektu stejným způsobem je zobrazena možnost vytvoření nového efektu. Ta umožní vybrat typ, nastavit jeho případné parametry a upravit počáteční a koncový čas, pokud je to potřeba. Po vytvoření je každý efekt znázorněn graficky. Uživatel není nijak omezen, efekty může vkládat libovolně, vzájemně je překrývat a kombinovat. Vložené efekty lze kdykoliv editovat nebo odstranit. V tom případě je potřeba provést dlouhý dotek na daném efektu. Pokud se jich v daném místě překrývá více, je zobrazen jejich seznam a lze vybrat konkrétní jeden.

Tlačítko "Export result" umožní spustit vytvoření výsledného video souboru.



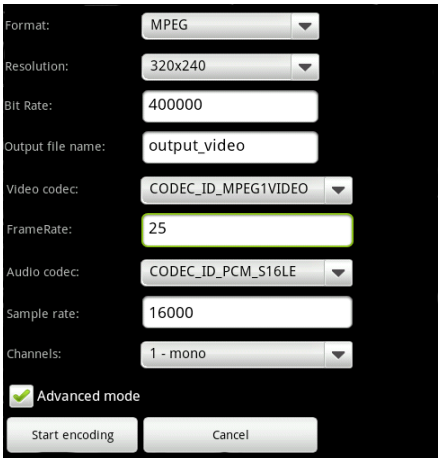
Obrázek 17. Časová osa s výsledným videem.

3.4.3 Výstupní parametry

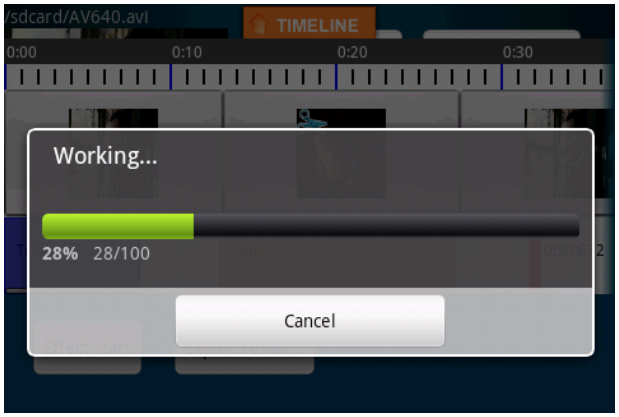
Před spuštěním kódování výsledného videa je možné nastavit jeho parametry. V **základním módu** je možné zvolit jeden ze šesti výstupních formátů s přednastavenými parametry. Jejich hodnoty jsou zobrazeny bez možnosti editace. Tyto nabízené předvolby byly testovány a ověřeny. Uživatel v tomto základním módu může ještě zvolit rozlišení, název výstupního souboru a požadovaný bitrate.

Pokud se uživatel přepne do **rozšířeného módu**, může veškeré parametry svobodně nastavovat (viz. obrázek 18). Lze zvolit formát, video a audio kodek (ze všech dostupných, knihovnou FFmpeg podporovaných), nastavit vzorkovací frekvenci i počet kanálů audia a nastavit počet snímků za sekundu ve výstupním videu. Kombinace parametrů není nikterak hlídána. Špatně zvolené parametry mohou způsobit nekorektní výsledný video soubor nebo také pád aplikace. Video kodeků je přes 150 a audio kodeků přibližně 100, otestovat všechny použitelné možnosti a hlídat nastavení parametrů aplikací by bylo extrémně náročné a složité.

Po spuštění kódování výsledného videa je zobrazen průběh s možností kódování zastavit (viz. obrázek 19). Při zastavení kódování uživatelem je formát i výstupní soubor korektně uzavřen a dokončen. Výsledné video pak zůstává v souborovém systému a lze ho běžně přehrát v jiném přehrávači. Toto se dá využít například při zpracování delšího videa pro získání několikasekundového vzorku výstupu. Poté co je uživatel s nastavením spokojen, může ponechat kódování dojít až do konce. O úspěšném dokončení je uživatel informován zprávou s celkovým časem trvání.



Obrázek 19. Rozšířený mód nastavení.



Obrázek 18. Průběh kódování.

3.5 Testování výkonnosti

Zpracování videa je značně výpočetně náročná úloha, kladoucí nároky na výkon hardwaru. V případě práce s videem na mobilních zařízeních proto přirozeně vzniká otázka, zdali jsou současná zařízení již schopna dostatečně a v rozumném čase tyto úlohy provádět.

Přehrávání videa je v současných zařízeních běžně a široce podporováno. Dekódování videa je totiž relativně méně náročné a funguje rychle a použitelně i na výkonnostně slabším, a tudíž i levnějším hardwaru. Pokud ovšem ale přijde na zpracování videa a jeho úpravy, nelze se vyhnout kódování videa, které už je značně náročným procesem. Jedním z cílů této práce je také ověřit, jak rychle jsou současná mobilní zařízení schopna právě kódování videa provádět. Tato podkapitola uvádí výsledky naměřené na reálných mobilních zařízeních běžně dostupných na trhu. Měření je prováděno přímo aplikací implementovanou v této diplomové práci.

3.5.1 Použitá zařízení

K testování rychlosti byly použity tři modely mobilních telefonů. Zástupcem těch levnějších, méně výkonných a dříve uvedených na trh je *HTC Hero*. Jakýmsi prostředním modelem je *HTC Desire* a zástupcem těch novějších *Nexus S*. Do následujícího přehledu (tabulky 3, 4 a 5) byly vybrány parametry, které jsou důležité pro zpracování videa, tedy procesor a velikost RAM. Dále informace o verzi systému a datum uvedení na trh, které přináší představu o jakou "generaci" telefonu jde. Navíc je uvedena přibližná aktuální cena, aby bylo možné přístroj zařadit také do cenové kategorie. Údaje v tabulkách odpovídají konkrétně použitým variantám přístrojů. Parametry byly zjištěny na webových stránkách výrobců.

Procesor	Qualcomm **528 MHz**
RAM	288 MB
Systém	Android 2.1
Uvedení na thr	červen 2009
Přibližná cena	6 000 Kč

Tabulka 3. Parametry HTC Hero.

Procesor	Qualcomm QSD8250 Snapdragon **1 GHz**
RAM	576 MB
Systém	Android 2.2
Uvedení na trh	Květen 2010
Přibližná cena	10 000 Kč

Tabulka 4. Parametry HTC Desire.

Procesor	Samsung Hummingbird **1 GHz**
RAM	512 MB
Systém	Android 2.3.4
Uvedení na trh	začátek 2011
Přibližná cena	12 000 Kč

Tabulka 5. Parametry Google Nexus S.

3.5.2 Naměřené hodnoty

Testování bylo zaměřeno na měření času překódování videa. V aplikaci bylo otevřeno zdrojové video a poté spuštěno překódování do zvoleného cílového formátu. Aby byl vliv doby dekódování vstupního videa a audia vždy přibližně stejný, vstupní video bylo vždy obsahově stejné s parametry, které jsou uvedeny v tabulce 6.

Formát	MP4
Video kodek	MPEG-4 Video (FVFW)
Snímků za sekundu	25
Audio kodek	PCM S16 LE (araw)
Vzorkovací frekvence	16 000 Hz
Kanály	Mono

Tabulka 6. Parametry vstupního videa.

Testovací video bylo převzorkováno do čtyř různých rozlišení se zachováním dříve zmíněných parametrů. Varianta 176×144 pixelů reprezentuje malá videa vhodná například jako

přílohy různých multimediálních zpráv, rozlišení 320×240 a 640×480 běžné standardní velikosti videí pořizovaných kamerami na mobilních zařízeních a poslední varianta 1320×720 je zástupcem HD rozlišení. Délka takto vzniklých vstupních video souborů byla vždy stejná, konkrétně 12 vteřin. Jako testované výstupní formáty byly zvoleny MPEG, MP4 a WMV, kterým byly nastaveny parametry uvedené v tabulce 7, s cílem použít různé audio a video kodeky.

Výstupní formát	**MPEG**	**MP4**	**WMV**
Video kodek	MPEG 1 Video	MPEG 4 Video	MS MPEG4 v3
Snímků za sekundu	25	25	25
Audio kodek	MP2	AAC	WMA v1
Vzorkovací frekvence	16 000 Hz	16 000 Hz	32 000 Hz
Kanály	Mono	Mono	Mono

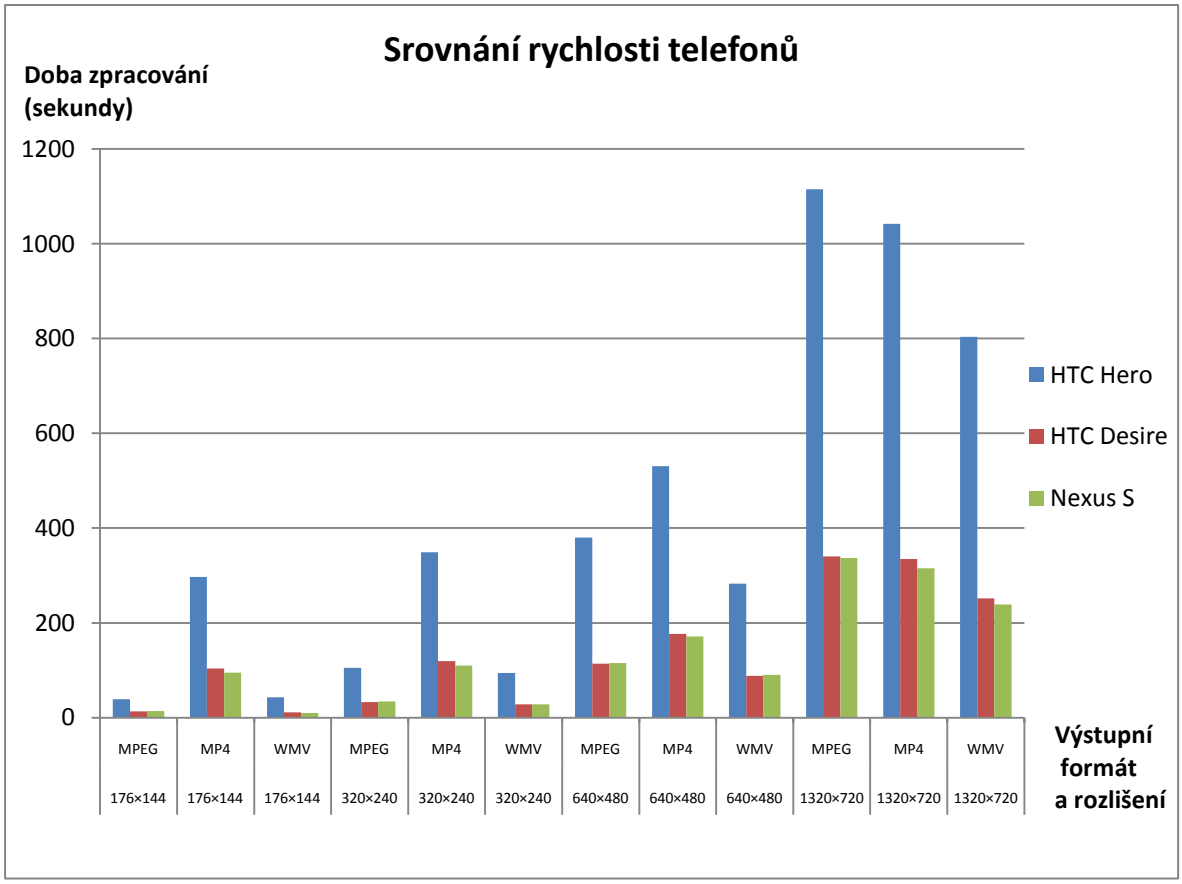
Tabulka 7. Výstupní testované formáty a jejich parametry.

Nejprve byly provedeny testy, kdy rozlišení vstupního videa je stejné jako rozlišení videa výstupního, tudíž bez vlivu převzorkování rozlišení obrazu. Všechny testy byly provedeny 3 krát a pokaždé byla měřena doba potřebná ke kompletnímu překódování obrazu i zvuku a korektnímu dokončení zápisu výstupního video souboru. Čas byl následně zprůměrován a zaokrouhlen na celé sekundy. Výsledné údaje jsou obsaženy v tabulce 8. Pro lepší přehled jsou data z tabulky vynesena do grafu 1, který srovnává časy jednotlivých telefonů pro jednotlivé formáty a rozlišení.

Vstupní rozlišení	**Výstupní rozlišení**	**Výstupní formát**	**HTC Hero**	**HTC Desire**	**Nexus S**
176×144	176×144	MPEG	0:39	0:13	0:14
176×144	176×144	MP4	4:57	1:44	1:35
176×144	176×144	WMV	0:43	0:11	0:10
320×240	320×240	MPEG	1:45	0:33	0:34
320×240	320×240	MP4	5:49	1:59	1:50
320×240	320×240	WMV	1:34	0:28	0:28
640×480	640×480	MPEG	6:20	1:54	1:55
640×480	640×480	MP4	8:51	2:57	2:51
640×480	640×480	WMV	4:43	1:28	1:30
1320×720	1320×720	MPEG	18:35	5:40	5:37
1320×720	1320×720	MP4	17:22	5:35	5:15
1320×720	1320×720	WMV	13:24	4:12	3:59

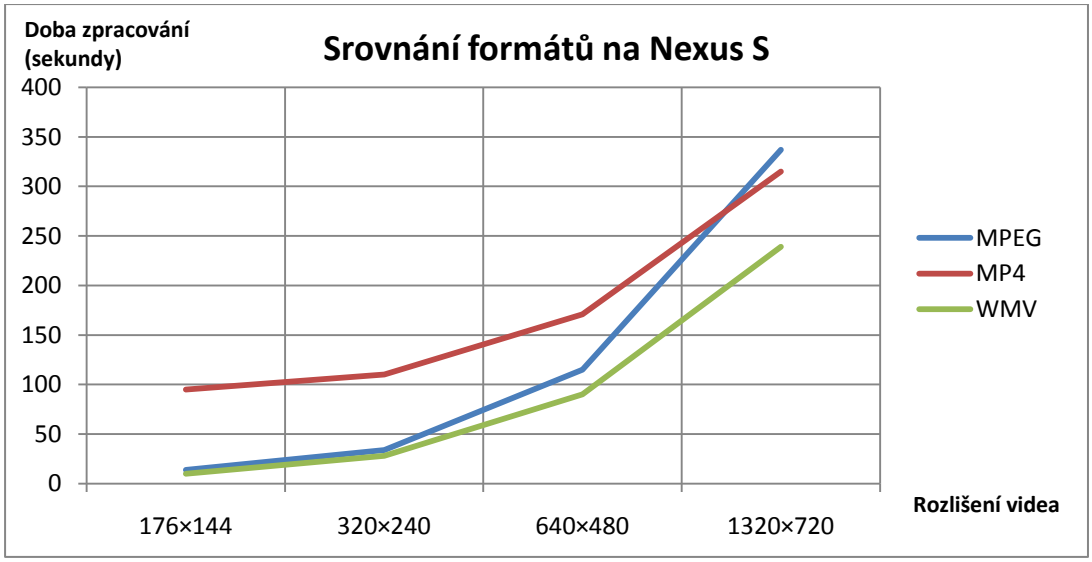
Tabulka 8. Naměřené časy doby překódování videí ze vstupního formátu do výstupního.

Poznámka: Uvedené časy jsou ve formátu minuty:sekundy, uvedená rozlišení udávají výšku×šířku obrazu v pixelech.

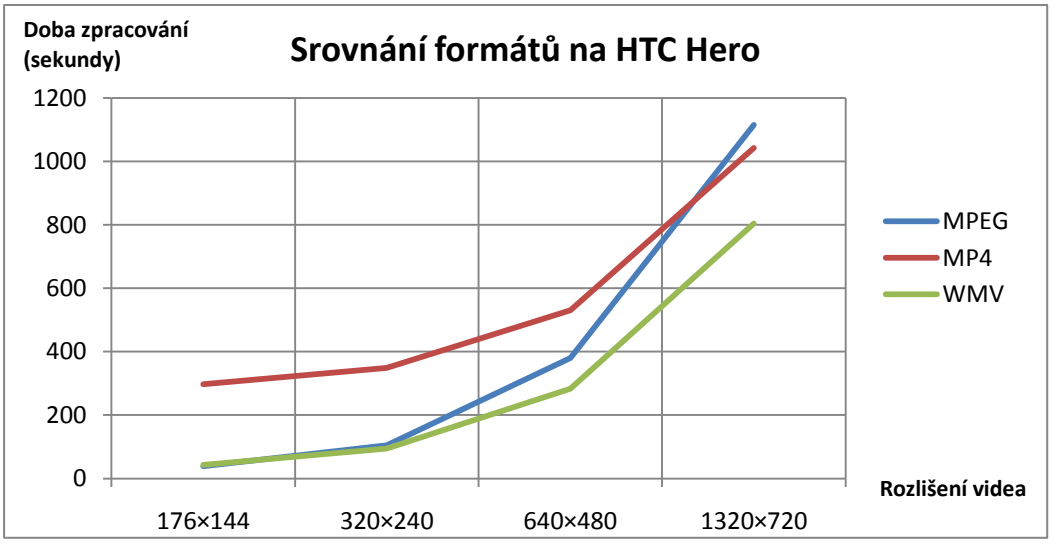


Graf 1. Srovnání doby potřebné pro překódování videa do různých výstupních formátů a rozlišení.

Zajímavé je taktéž srovnat jednotlivé formáty mezi sebou. Následující grafy 2 a 3 zobrazují nárůst potřebného času zpracování u jednotlivých formátů s rostoucím rozlišením. Pokud se podíváme na naměřené časy zpracování pro telefon *Nexus S* (který má velmi podobné hodnoty jako *HTC Desire*) a *HTC Hero* zjistíme, že průběh nárůstu potřebného času je de facto stejný. Celkový čas je ovšem pochopitelně větší na hardwarově slabším *HTC Hero*.



Graf 2. Srovnání doby zpracování pro jednotlivé výstupní formáty na Nexus S.



Graf 3. Srovnání doby zpracování pro jednotlivé výstupní formáty na HTC Hero.

V předchozích testech bylo prováděno testování bez změny rozlišení videa. Tato operace má zcela jistě vliv na celkovou dobu zpracování. Proto byly provedeny následující testy s cílem zjistit, jak velký podíl má na celkovém času právě převzorkování rozlišení obrazu. Opět byla použita stejná vstupní videa a postupy jako v předchozích testech, s rozdílem, že výstupní video mělo jiné rozlišení a proto při zpracování každého snímku bylo provedeno jeho převzorkování. Bylo testováno jak převzorkování malého videa na větší rozlišení (ze 176×144 pixelů na 640×480), tak opačná varianta, zmenšení většího videa na menší (ze 640×480 pixelů na 176×144). Zjištěné hodnoty uvádí následující tabulka 9.

Vstupní rozlišení	Výstupní rozlišení	Výstupní formát	HTC Hero	HTC Desire	Nexus S
176×144	640×480	MPEG	8:02	2:29	2:32
176×144	640×480	MP4	10:32	3:30	3:26
176×144	640×480	WMV	6:27	2:01	2:06
640×480	176×144	MPEG	3:01	0:54	1:14
640×480	176×144	MP4	7:18	2:25	2:37
640×480	176×144	WMV	3:05	0:53	1:10

Tabulka 9. Naměřené časy doby překódování videí s převzorkováním rozlišení.

Samotné hodnoty doby trvání převzorkování ovšem získáme až odečtením času překódování videa ve stejném rozlišení (tento čas byl zjištěn v předchozí sadě testů). Následující tabulka 10 obsahuje počet sekund, o kolik bylo prodlouženo zpracování videa kvůli nutnosti převzorkování. Jistou nepřesnost může zanést fakt, že dekódování videa v menším rozlišení je pochopitelně rychlejší. Nicméně tento rozdíl je velmi malý a zkresluje tedy výsledky jen nepodstatně.

Vstupní rozlišení	Výstupní rozlišení	Výstupní formát	HTC Hero	HTC Desire	Nexus S
176×144	640×480	MPEG	102 s	35 s	37 s
176×144	640×480	MP4	101 s	33 s	35 s
176×144	640×480	WMV	104 s	33 s	36 s
640×480	176×144	MPEG	142 s	41 s	60 s
640×480	176×144	MP4	141 s	41 s	62 s
640×480	176×144	WMV	142 s	42 s	60 s

Tabulka 10. Doba, o kolik se prodlouží čas zpracování navíc se změnou rozlišení obrazu.

Posledním testovaným údajem je vliv použití obrazových efektů na prodloužení času zpracovávání videa. Opět byla použita videa stejná jako v předchozích testech. Byla vybrána dvě různá rozlišení a jako reprezentant efektů převod do šedotónového obrazu. Jeho složitost je přibližně stejná jako prahování i sépiový filtr. Na videa byl tedy aplikován převod do šedotónového obrazu v celé jejich délce od začátku do konce. Naměřené celkové časy zpracování úvádí následující tabulka 11.

Vstupní rozlišení	Výstupní rozlišení	Výstupní formát	HTC Hero	HTC Desire	Nexus S
176×144	176×144	MPEG	1:10	0:23	0:27
176×144	176×144	MP4	5:27	1:52	2:18
176×144	176×144	WMV	1:23	0:22	0:21
320×240	320×240	MPEG	3:19	1:02	1:12
320×240	320×240	MP4	7:07	2:31	2:23
320×240	320×240	WMV	3:00	0:58	1:04

Tabulka 11. Naměřené časy doby překódování videí s aplikovaným šedotónovým efektem.

Stejně jako v případě předchozího testu (vliv převzorkování) je třeba odečíst čas kódování videa v odpovídajícím rozlišení. Následující tabulka 12 pak obsahuje počet sekund, o které se prodloužila doba zpracování videa kvůli použití obrazového efektu.

Vstupní rozlišení	Výstupní rozlišení	Výstupní formát	HTC Hero	HTC Desire	Nexus S
176×144	176×144	MPEG	31 s	10 s	13 s
176×144	176×144	MP4	30 s	9 s	14 s
176×144	176×144	WMV	33 s	11 s	13 s
320×240	320×240	MPEG	85 s	30 s	38 s
320×240	320×240	MP4	84 s	32 s	36 s
320×240	320×240	WMV	86 s	30 s	36 s

Tabulka 12. Doba, o kolik se prodlouží čas zpracování navíc s aplikovaným šedotónovým efektem.

3.5.3 Zhodnocení naměřených výsledků

Porovnáme-li výsledky telefonů, není překvapením, že hlavní roli hraje rychlost procesoru. Doba zpracování videí je na telefonech *HTC Desire* a *Nexus S* přibližné stejná, liší se občas pouze v řádu několika málo sekund. Naměřené doby jsou přibližně třetinové oproti *HTC Hero*. Roli hraje zcela jistě jeho pomalejší procesor s poloviční taktováním. Dalšími faktory mohou být menší velikost dostupné paměti, starší verze systému, případně i rychlost zápisu do flash paměti, která je u novějších zařízení zcela jistě rychlejší.

Podíváme-li se na výsledky z pohledu praktické použitelnosti, můžeme konstatovat, že úpravy kratších videoklipů v menších rozlišeních je možné v rozumném čase provádět na všech testovaných zařízeních. Sem spadá například editace a následné exportování videa s malým rozlišením vhodným pro posílání pomocí multimediálních zpráv. Na rychlých telefonech při vhodných výstupních formátech potom vychází kódování 1 minuty videa na 1 minutu práce (v malém rozlišení 176×144). To je zcela jistě velmi příznivý a prakticky použitelný výsledek. Ve větším rozlišení 320×240 jsou výsledky na novějších telefonech taktéž dobré, zpracování 1 minuty videa trvá přibližně 2,5 minuty.

Zpracování 1 minuty videa ve středním rozlišení 640×480 trvá přibližně 10 minut na rychlejších telefonech (při volbě vhodného formátu). Zde už si lze klást otázku, zdali je takovýto čas ještě přijatelný. Nejspíše bude záležet na konkrétní situaci, trpělivosti uživatele a stavu baterie zařízení. Lze si představit situaci, kdy je uživatel třeba v hotelu na dovolené, pořídí záběry mobilním telefonem a potřebuje sestavit videoklip a někde ho publikovat nebo někomu odeslat. Pokud má výsledné video řádově minuty až desítky minut a je k dispozici zdroj elektrické energie, může být zpracování spuštěno večer a ráno má uživatel k dispozici výsledek.

V případě videa ve vysokém rozlišení 1320×720 trvá kódování 1 minuty videa přibližně 25 minut (podle formátu) i na rychlejších telefonech, což už je skutečně velmi vysoká hodnota. Kódování videí s takto vysokým rozlišením je stále ještě pro mobilní zařízení náročným úkolem a nelze se asi představit praktické použití, kromě zpracování velmi krátkých klipů nebo použití v nějakých velmi nutných situacích. Na pomalejších telefonech jako *HTC Hero* trvá kódování 1 minuty videa v tomto rozlišení více než hodinu a čtvrt. Můžeme tak tvrdit, že toto je prakticky nepoužitelná hodnota.

Podíváme-li se na jednotlivé testované formáty, průběh nárůstu času potřebného pro zpracování s rostoucím rozlišením je dle očekávání stejný na všech testovaných telefonech. Jednoznačně nejrychleji probíhalo kódování do formátu WMV. To trvalo nejkratší dobu při všech testovaných rozlišeních. Formát MPEG dosahoval podobných časů při nižších rozlišeních, ale jeho nárůst času byl strmější a v případě rozlišení 1320×720 dokonce přesáhl časy obou dalších formátů. Zajímavý průběh byl zjištěn u formátu MP4. Čas potřebný pro kódování videa v nejnižším rozlišení 176×144 byl až 3 krát větší než u zbylých formátů. V případě dalších rozlišení byl asi dvojnásobný, nicméně v případě rozlišení 1320×720 se již ale velmi blíží času WMV a dokonce se dostává pod čas formátu MPEG. Z výsledků tedy vyplývá, že s výjimkou vysokých rozlišení bude vždy kódování do formátu MP4 trvat minimálně dvojnásobnou dobu, zatímco doby zpracování MPEG a WMV jsou velmi podobné.

V dalších testech byl zjišťován vliv převzorkování obrazu na dobu zpracování. Po odečtení časů zpracování videa ve stejném rozlišení byly získány doby nárůstu potřebného času (tabulka 10). Samotná operace převzorkování je nezávislá na výstupním formátu, proto jsou hodnoty pro daný telefon a rozlišení přibližně stejné pro všechny formáty, rozdíly v řádu jednotek jsou způsobeny zaokrouhlovacími chybami. Zjištěné údaje naznačují, že operace změny rozlišení obrazu je z pohledu celkové doby zpracování nezanedbatelnou položkou. Například při výstupním formátu MPEG z celkového času kódování zabírá přibližně 22 procent času na všech typech testovaných telefonů při výstupním rozlišení 640×480. Zajímavá je situace u nižších rozlišení, kde operace převzorkování

obrazu trvá mnohonásobně déle než samotné kódování. Příkladem může být výstupní rozlišení 176×144 ve formátu MPEG, kde na telefonu *HTC Desire* převzorkování zabírá až 75 procent celkového času. Z údajů lze také odvodit, že s rostoucím rozlišením klesá poměr doby převzorkování na celkové době zpracování videa.

Poslední sada testů zjišťovala vliv použití obrazových efektů na prodloužení celkové doby zpracování. Taktéž se nejedná o zanedbatelný podíl, protože se provádí výpočetní operace nad každým pixelem snímku a také konverze barevného prostoru (dekódované snímky z videa jsou ve formátu YUV, efekty pracují v RGB formátu). Nárůst doby zpracování v sekundách použitím šedotónového efektu obsahuje tabulka 12. Z tabulky vyplývá, že i přidání efektu se významně podílí na celkové době zpracování. Podobně jako u převzorkování, čím nižší rozlišení, tím větší podíl. Například u telefonu *HTC Desire* při rozlišení 176×144 a výstupním formátu MPEG je celkový čas prodloužen až dvojnásobně.

3.5.4 Srovnání s existujícími aplikacemi

Pokud porovnáme aplikaci s dříve uvedenými v kapitole 2.1.2, můžeme vyvodit následující závěry. Z pohledu funkcionality nabízí aplikace v podstatě podobné možnosti jako *Clesh Video Editor* (spojování různých klipů, střih, efekty, podpora různých formátů). Na rozdíl od něj se však kódování videa provádí přímo na mobilním zařízení a neodesílá zpracovávaná videa na vzdálený server. Umožňuje tak provádět úpravy bez nutnosti síťového připojení. Jistou nevýhodou je samozřejmě nárok na výkon mobilního zařízení a spotřebu energie.

Další aplikace *VidTrim* a *Snip Video Trimmer* umožňují pouze zkracování existujících videí a nikoliv jejich spojování či překódování. Překódování do jiného formátu nabízí pouze *VidTrim* v placené verzi, ale umožňuje pouze export do formátu MP4. Poslední zmíněná aplikace *Videocam Illusion* neumožňuje žádné úpravy videa, pouze přidávat obrazové efekty při natáčení videa pomocí kamery mobilního zařízení.

Video editor pro Android vytvořený v této práci tak přináší další nové možnosti pro práci s videem na této platformě. Žádná jiná aplikace doposud také nenabízí možnost přímo na zařízení převádět videa mezi tak širokou škálou formátů a kodeků.

4 Závěr

Cílem této práce bylo implementovat jednoduchou aplikaci pro editaci videa na mobilní platformě Android. Práce v první části poskytla potřebné teoretické informace pro zasazení problematiky do širšího kontextu.

Nejprve přinesla přehled dostupných nástrojů a možností na běžných desktopových počítačích. Potom byly zmíněny existující aplikace pro editaci videa přímo pro Android, včetně jejich základních vlastností. V další podkapitole byla popsána samotná platforma, zmíněna historie vývoje a popsána její struktura a architektura. Dále se text zaměřuje na vývoj aplikací, hlavně těch v nativním kódu. Část aplikace je implementována v jazyce Java a část v jazyce C. Provázání těchto částí zajišťuje rozhraní Java Native Interface, proto je mu taktéž věnována pozornost. Při implementaci byla využita multimediální knihovna FFmpeg, proto jí byla věnována jedna z podkapitol. Poskytuje základní informace o jejích vlastnostech a podporovaných formátech a kodecích.

Druhá hlavní část práce se věnuje už samotné aplikaci. Popisuje její návrh, strukturu a postup implementace. Zaměřuje se především na podstatné části a řešení důležitých problémů. Popsáno je také výsledné uživatelské rozhraní a princip jeho ovládání. Po implementaci aplikace bylo možné využít ji a vyzkoušet tak rychlost zpracování videa na současných mobilních telefonech. Proto je jedna podkapitola věnována výkonnostním testům. Snahou bylo zjistit, jak dlouho bude zpracování videa trvat na různých telefonech a zjistit, zdali je tato úloha na mobilních zařízeních vůbec prakticky použitelná. Výsledky testů jsou poskytnuty ve formě tabulek a grafů a následně zhodnoceny.

V závěru se práce věnuje srovnání implementované aplikace s těmi již existujícími. Ukazuje se, že aplikace přináší nové možnosti ve zpracování videa na platformě Android. Jde především o kódování videa přímo na mobilním zařízení a možnost použití široké škály vstupních i výstupních formátů a kodeků. Co se týče možností dalšího vývoje projektu, lze určitě dále aplikaci rozšiřovat. Především se nabízí možnost implementace dalších obrazových efektů a zlepšování uživatelského rozhraní. Další možností rozšíření funkcionality by také mohla být možnost nezávisle upravovat zvukovou stopu, umožnit otevírání čistě zvukových klipů a nahrazovat jimi původní zvuk videa.

Tato diplomová práce tedy ukázala, že v současné době lze už i na mobilních zařízeních použitelně zpracovávat video. To je možné nejen díky stále rychlejšímu hardwaru, ale také díky novým platformám jako Android, které poskytují vývojářům mobilních aplikací bohaté a rozmanité možnosti.

Literatura

[1] Wikipedia: *Comparison of video editing software*. [online]. [cit. 2011-03-20]. Dostupný z WWW:
< http://en.wikipedia.org/wiki/Comparison_of_video_editing_software>.

[2] Adobe Systems: *Adobe Premiere Pro CS5.5 / Features.* [online]. [cit. 2011-03-20]. Dostupný z WWW:
< http://www.adobe.com/cz/products/premiere/features.html>.

[3] Microsoft Corporation: *Windows Live Movie Maker 2011*. [online]. [cit. 2011-03-20]. Dostupný z WWW:
< http://explore.live.com/windows-live-movie-maker?os=other>.

[4] *Free and open source video editor for GNU/Linux, Mac OS X and FreeBSD*. [online]. [cit. 2011-03-21]. Dostupný z WWW:
< http://www.kdenlive.org/features>.

[5] Google: *Android Market.* [online]. [cit. 2011-04-12]. Dostupný z WWW:
< https://market.android.com/>.

[6] *AppBrain App Market.* [online]. [cit. 2011-04-12]. Dostupný z WWW:
< http://www.appbrain.com/>.

[7] Google: *Developer Guide - What is Android.* [online]. [cit. 2010-12-10]. Dostupný z WWW:
<http://developer.android.com/guide/basics/what-is-android.html>.

[8] Wikipedia: *Android (operating system).* [online]. [cit. 2010-11-20]. Dostupný z WWW:
<http://en.wikipedia.org/wiki/Android_%28operating_system%29>.

[9] TechRadar – online magazíne: *A complete history of Android*. [online]. [cit. 2010-12-16]. Dostupný z WWW:
<http://www.techradar.com/news/phone-and-communications/mobile-phones/a-complete-history-of-android-470327>.

[10] Google: *Android Architecture scheme.* [online]. [cit. 2010-12-10]. Dostupný z WWW:
<http://developer.android.com/images/system-architecture.jpg>.

[11] Google: *Installing SDK.* [online]. [cit. 2010-12-10]. Dostupný z WWW:
<http://developer.android.com/sdk/index.html>.

[12] Mlích, Jozef: *Tvorba aplikací pro mobilní zařízení, přednáška Android*. [online]. Vysoké učení technické, Brno, aktualizováno 2009-11-30.

[13] Google: *What is NDK?.* [online]. [cit. 2010-12-10]. Dostupný z WWW:
< http://developer.android.com/sdk/ndk/overview.html>.

[14] Google: *Installing the NDK.* [online]. [cit. 2011-04-18]. Dostupný z WWW: < http://developer.android.com/sdk/ndk/index.html>.

[15] Sun Microsystems: *The Java Native Interface: Programmer's Guide and Specification-Basic Types, Strings, and Arrays.* [online]. [cit. 2011-04-21]. Dostupný z WWW: < http://java.sun.com/docs/books/jni/html/objtypes.html#5279>.

[16] Sun Microsystems: *The Java Native Interface: Programmer's Guide and Specification - JNI Types.* [online]. [cit. 2011-04-20]. Dostupný z WWW: < http://java.sun.com/docs/books/jni/html/types.html >.

[17] Google: *The AndroidManifest.xml File*. [online]. [cit. 2011-04-20]. Dostupný z WWW: < http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

[18] Night Dreaming (by Sudar)*: The structure of an Android project*. [online]. [cit. 2011-05-20]. Dostupný z WWW: < http://sudarmuthu.com/blog/the-structure-of-an-android-project>.

[19] *About FFmpeg.* [online]. [cit. 2010-11-30]. Dostupný z WWW: <http://www.ffmpeg.org/about.html>.

[20] Wikipedia: *FFmpeg*. [online]. [cit. 2010-11-30]. Dostupný z WWW: <http://en.wikipedia.org/wiki/FFmpeg>.

[21] Wikipedia:. *libavcodec.* [online]. [cit. 2010-12-2]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Libavcodec>.

[22] Kršek, P., Španěl, M.: *Základy počítačové grafiky, slajdy k přednášce Redukce barevného prostoru.* Vysoké učení technické, Brno, 2011.

[23] Smith, Zach: *Convert images to grayscale and sepia*. [online]. [cit. 2011-05-12]. Dostupný z WWW: < http://www.techrepublic.com/blog/howdoi/how-do-i-convert-images-to-grayscale-and-sepia-tone-using-c/120>.

Seznam příloh

Příloha 1: CD se zdrojovými soubory.

Obsah CD

Přiložené CD obsahuje následující části:

- *doc* – technická zpráva v elektronické podobě
- *image* – plakát
- *package* – instalační balíček aplikace
- *src* – projekt pro Eclipse IDE se zdrojovými soubory aplikace
- *video* – demonstrační video