České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Pokročilý záznamník videa s hlasovým ovládáním na mobilním telefonu

Bc. Tomáš Buzek

Vedoucí práce: prof. Ing. Jiří Matas, Ph.D.

Studijní program: Otevřená informatika, Magisterský
Obor: Softwarové inženýrství
12. května 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Buzek**

Studijní program: Otevřená informatika (magisterský)
Obor: Softwarové inženýrství

Název tématu: **Pokročilý záznamník videa s hlasovým ovládáním na mobilním telefonu**

Pokyny pro vypracování:

Cílem práce je upravit existující mobilní aplikaci Zoomera pro operační systém Android, která s využitím kamery na přístroji funguje jako lupa a jednoduché záznamové zařízení, tak aby umožňovala ovládání hlasem a možnost nahrávání videa v nekonečné smyčce.

1. Seznamte se s možnostmi nahrávání videa na zařízeních Android.
2. Zakomponujte do aplikace nahrávání obrazu z kamery fotoaparátu.
3. Definujte vhodný způsob nahrávání videa v nekonečné smyčce.
4. Implementujte nahrávání v nekonečné smyčce.
5. Ze semestrálního projektu převezměte metodu ovládání hlasem. Nalezněte takové nastavení knihovny, které bude mít pro uživatele největší přínos.
6. V případě nevyhovující funkce vyberte jinou metodu nebo navrhněte úpravy.
7. Otestujte hlasové ovládání s více uživateli s vestavěným mikrofonem, externím mikrofonem, popřípadě bluetooth mikrofonem.
8. Řešte sladění výpočetních a paměťových nároků implementovaných součástí, tak aby byla zajištěna maximální funkčnost aplikace.

Seznam odborné literatury:

[1] Develop | Android Developers. GOOGLE INC. [online].
Dostupné z: http://developer.android.com/develop/index.html
[2] CMU Sphinx - Speech Recognition Toolkit | Carnegie Mellon University. [online].
Dostupné z: http://cmusphinx.sourceforge.net/
[3] HUANG, Xuedong. Spoken language processing: a guide to theory, algorithm, and system development.
Vyd. 1. New Jersey: Prentice-Hall, 2001. ISBN 01-302-2616-5.

Vedoucí: prof.Ing. Jiří Matas, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 19. 2. 2014

Poděkování

Tímto bych v první řadě rád poděkoval panu prof. Ing. Jiřímu Matasovi, Ph.D. za vedení této práce, za veškeré konzultace, nové nápady a připomínky, které byly pro tuto práci velmi přínosné, a za čas, který mi věnoval. Dále bych rád poděkoval svým nejbližším, kteří na mě nezapomněli a podporovali mě i ve chvílích, kdy jsem velkou část svého času a úsilí věnoval právě této práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Ve Strančicích dne 12. května 2014 .....

Abstrakt

Tato diplomová práce pojednává o vytvoření mobilní aplikace pro systém Android, která slouží jako pokročilé záznamové video zařízení s hlasovým ovládáním. Popisuje metody nahrávání z kamery zařízení spolu s jejich implementací, diskutuje jejich výhody a nevýhody. Pokrývá nestandardní možnosti záznamu videa, jako je zpracování nahrávaného obrazu v reálném čase na GPU zařízení pomocí OpenGL ES, které umožňuje aplikaci obrazových filtrů, velkých digitálních zvětšení obrazu a přidání grafických prvků přímo do videa. Zpracování jednotlivých snímků videa probíhá ihned při jeho záznamu, toho využívá funkce nahrávání videa v nekonečné smyčce. Ta umožňuje nepřetržité nahrávání obrazu z kamery zařízení tak, že uloženo je vždy pouze posledních několik časových jednotek záznamu. Obsahem jsou výsledky měření metod nahrávání videa, které potvrzují, že implementované metody nahrávání fungují a jsou použitelné v reálných podmínkách. Práce obsahuje vyhodnocení řady open source nástrojů pro rozpoznávání řeči, kde byla vybrána knihovna CMU Sphinx a použita v modulu hlasového ovládání aplikace. Princip pro kontinuální rozpoznávání řeči je založen na použití aktivačního slova a konkrétního příkazu. Hlasové ovládání bylo testováno s uživateli ve věku od 20 do 60 let a je ověřeno, že je funkční a použitelné pro ovládání zařízení. Aplikace je vybavena grafickým uživatelským rozhraním, které umožňuje kontrolu nad všemi implementovanými součástmi.

Klíčová slova

Nahrávání videa, záznam v nekonečné smyčce, hlasové ovládání, rozpoznávání řeči, zpracování obrazu, mobilní zařízení, Android, OpenGL, CMU Sphinx

Abstract

This thesis describes the development of a mobile application for Android system which can be used as an advanced voice-controlled video recorder. It describes recording methods that use the device's camera including their implementations, discusses advantages and disadvantages. It contains special possibilities of video recording as real time GPU processing of recorderd frames using OpenGL which allows application of image filters, large digital zoom and adding graphical elements directly in video. Processing of every single frame is done immediately during video recording, it is used for loop video recording. That allows to record video from camera continuously while only last few time units are stored. Included results of measurements confirms that recording methods are functional and could be used in real conditions. This thesis contains evaluation of open source speech recognition tools where CMU Sphinx library was chosen and used in application's voice control module. The principle of continual speech recognition is based on usage of activation word together with particular command. Voice control was tested with users from 20 to 60 years old, its functionality has been proved and it can be used for application controlling. Application is equipped with graphical user interface which allows to control all implemented modules.

Keywords

Video recording, infinite loop recording, voice control, speech recognition, image processing, mobile device, Android, OpenGL, CMU Sphinx

Obsah

Zkratky

V práci byly použity následující zkratky.

2D	dvourozměrný
3D	trojrozměrný
AVC	Advanced Video Coding
BSD	Berkeley Software Distribution
CMU	Carnegie Mellon University
CPU	Central Processing Unit, Centrální výpočetní jednotka
DCT	Discrete Cosine Transform, Diskrétní kosinová transformace
FPS	Frames per second, Snímky za vteřinu
GUI	Graphical User Interface, Grafické uživatelské rozhraní
HMM	Hidden Markov model, Skrytý Markovův model
JPEG	Joint Photographic Experts Group
LED	Light Emitting Diode, Dioda emitující světlo
Mbps	Megabit per second, Megabit za sekundu
MFCC	Mel-frequency cepstral coefficients, Melovské kepstrální koeficienty
MPEG	Moving Picture Experts Group
OpenGL	Open Graphics Library
OpenGL ES	OpenGL for Embedded Systems
PCM	Pulse-code modulation, Pulzně kódová modulace
SDK	Software Development Kit, Balík pro vývoj software
WAVE	Waveform Audio File Format

Cizojazyčné termíny

V práci byly použity následující cizojazyčné termíny.

Autofocus Funkce fotoaparátu/kamery umožňující automatické nastavení optiky za účelem dosažení ostrého obrazu.

Bluetooth Technologie bezdrátového přenosu informací mezi elektronickými zařízeními.

Buffer Vyrovnávací paměť pro dočasné uchování dat.

Callback Fragment kódu nebo odkaz na metodu, který je předán jako argument jiné metodě/třídě, u kterého se předpokládá, že bude zavolán po dokončení konkrétní akce nebo při konkrétní události.

Dekódování Proces dekomprese videa.

Enkódování Proces komprese videa.

Handsfree Zařízení, které umožňuje uživateli telefonovat tak, aby měl volné ruce.

Hardware Fyzické technické vybavení výpočetního zařízení.

Headset Zařízení kombinující sluchátka a mikrofón, přichytí se na hlavu uživatele.

Renderování Vytvoření obrazu pomocí počítačového programu na základě modelu.

Shader Jednoduchý program definovaný pro grafickou knihovnu OpenGL určený pro vykonávání na grafickém čipu.

1. Úvod

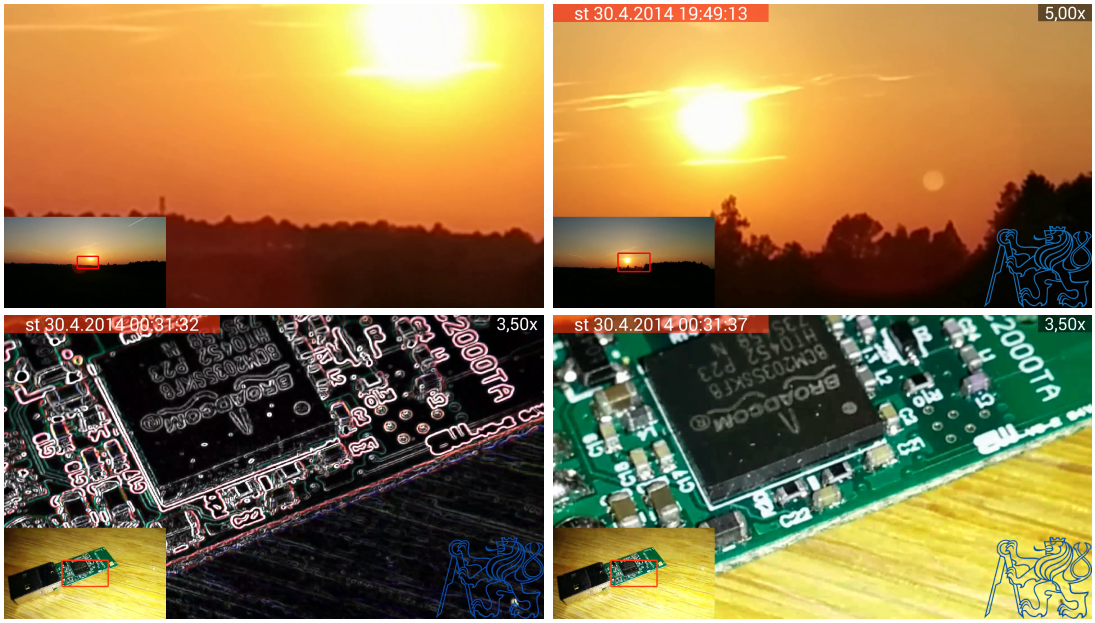
Využití potenciálu mobilních zařízení umožňuje nahrazení jinak drahých specializovaných přístrojů, relativně levnými zařízeními v kombinaci s adekvátní softwarovou výbavou. Výhodou takového použití je i fakt, že je má mnoho lidí pro každodenní použití vždy u sebe. Oproti úzce zaměřeným výrobkům, které jsou finančně nákladné a jsou nejčastěji určené pouze na použití při jedné konkrétní činnosti, nabízejí mobilní přístroje univerzální pole použití, kdy pro jeden konkrétní případ je postačující vytvořit či upravit příslušný software, který může propojit a řídit spolupráci mnoha součásti zařízení pro dosažení potřebné funkce.

Tato práce pokrývá vytvoření záznamového nástroje, který umožňuje uživateli provádět jeho ovládání hlasovými povely, a obsahuje pokročilé metody pro tvorbu video záznamu za použití kamery zařízení. Práce je rozdělena na několik kapitol, kde každá se věnuje odlišné problematice. V Kapitole 2 je vyhodnocen vzorek dostupných záznamových aplikaci pro systém Android. Následuje popis aplikace Zoomera 2012 v Kapitole 3, která byla základem pro implementaci modulů uvedených v této práci a byla vytvořena v předchozích projektech. Umožňuje ovládání základních součástí kamerového hardwaru a provádí základní zpracování obrazu na grafickém čipu. V Kapitole 4 jsou popsány technologie, nástroje a zařízení použité v této práci.

Následující část, Kapitola 5, popisuje implementaci modulu aplikace s pokročilými záznamovými funkcemi, které nejsou nikde na trhu k dispozici a kterým se nikdo v této kombinaci nevěnoval. Vzniká tak univerzální záznamník videa s přidanou hodnotou možnosti aplikovat vysoký digitální zoom, přidávat do obrazu efekty a údaje rozšiřující škálu možných informací, které výsledná nahrávka ponese. Výhodou oproti jiným podobným aplikacím tak je možnost zpracovávání obrazu na GPU zařízení v průběhu jeho nahrávání (pro příklad viz Obrázek 1). Kapitola obsahuje veškeré principy potřebné k vytvoření takového systému, počínaje od teorie digitálního videa, jeho komprese, způsobu ukládání, až po metody zpracování obrazu, nahrávání a jejich implementace.

Potenciál, který nabízí zpracování nahrávaných obrazových snímků, využívá metoda nahrávání videa v nekonečné smyčce v Kapitole 5.5. Je založena na nepřetržitém nahrávání obrazu z kamery zařízení, kdy je v každém okamžiku uloženo pouze posledních n časových jednotek záznamu. Je tak možné provádět kontinuální záznam obrazu, aniž dojde k zaplnění paměti zařízení. Využití může mít při zaznamenávání jakéhokoli náhodného jevu, u kterého nevíme, kdy se přesně vyskytne. Nepotřebujeme tak celý záznam, ale jen poslední časový úsek, který trvale uložíme ve chvíli, kdy k jevu dojde.

V Kapitole 6 je obsažen popis implementace funkce kontinuálního ovládání hlasem, které dovoluje uživateli věnovat se jiným činnostem a zároveň aplikaci plně ovládat. Příkladem je použití aplikace v laboratořích jako nástroje na kontrolu kvality a záznamu lidské činnosti. Uživatel se věnuje požadované činnosti a může aplikaci ovládat pouze hlasem, není třeba dotyku, a i tak nad ní neztrácí kontrolu. To je výhodné hlavně v kontaminovaných prostředích, kde by jinak nebylo možné se zařízením interagovat. Pro účely vytvoření tohoto modulu bylo nutné zahrnout základní principy zpracování zvukového signálu, strukturu lidské řeči a teorii rozpoznávání řeči. Je uvedeno vyhodnocení open source nástrojů pro rozpoznávání řeči, ze kterých byla použita knihovna CMU Sphinx pro implementaci hlasového ovládání.



Obrázek 1. Snímky z videa zaznamenaného aplikací vytvořenou v této práci demonstrují možnosti zpracování nahrávaného obrazu na GPU.

Kapitola 7 propojuje implementované moduly s uživatelem pomocí grafického uživatelského rozhraní. To umožňuje pohodlné a přehledné ovládání dotykem všech vytvořených součástí aplikace. Tím, že většinu jeho plochy pokrývá živý obraz z kamery zařízení, je zdůrazněna nejdůležitější část při použití aplikace, což je pozorovaná scéna.

Poslední část práce, Kapitola 8, uvádí výsledky testování jednotlivých vytvořených modulů. Metody nahrávání videa byly podrobeny množství měření, která určila kvalitu jejich návrhu a množství potřebného výpočetního výkonu pro jejich použití. Pro ověření kvality hlasového rozpoznávacího modulu a implementovaného hlasového ovládání bylo provedeno testování s uživateli ve věku od 20 do 60 let.

Systém Android je mobilní operační systém poskytovaný jako open source. Jedná se o složitý komplex, jehož vývoj neustále reaguje na nové technologie a zařízení a jež je rozšiřován o nové funkce. Z toho důvodu není triviální vyvíjet pro tuto platformu aplikace, zvlášť při využití hardwarového vybavení zařízení, jako je kamera. Mnoho funkcí má pouze základní dokumentaci, která však nijak detailně nepopisuje způsob jejich použití a interakce s ostatními součástmi systému. Jednou z částí systému, které se tento fakt týká, je zpracování obrazu z kamery, převážně v kombinaci se zpracováním na GPU pomocí OpenGL ES. Tyto technologie byly použity v této práci, která tímto může sloužit jako pokročilá dokumentace uvedených metod.

2. Existující aplikace

Tato kapitola zahrnuje existující implementace aplikací pro systém Android, jejichž účelem je pořizování záznamů pomocí vestavěné kamery mobilního zařízení. Všechny uvedené aplikace se nachází v internetovém obchodě Google Play[1].

První je referenční aplikace *Fotoaparát* od Google Inc.. Dalších šest bylo vybráno z kategorie *Fotografie* ve výše uvedeném obchodě. Jednalo se o aplikace, které patřily k nejstahovanějším, a zároveň byly posuzovány z hlediska přítomnosti těchto funkcí: nahrávání videa, hlasové ovládání, úpravy obrazu v reálném čase a vysoký digitální zoom. Uvedené aplikace byly vyzkoušeny na zařízení LG Nexus 5 (viz Kapitola 4.3.1).

2.1. Fotoaparát Google verze 2.1.043

Fotoaparát[2], jehož autorem je *Google Inc.*, je výchozí aplikací systému Android verze 4.4 pro pořizování fotografií a videozáznamů. Disponuje jednoduchým grafickým uživatelským rozhraním, jehož hlavním ovládacím prvkem je tlačítko spouště. Ostatní ovládací prvky se zobrazí tažením zleva doprava na displeji.

Základní funkce Aplikace umožňuje pořizování fotografií do formátu JPEG, nahrávání videí až v rozlišení 1920×1080, tvorbu sférických a panoramatických fotografií a disponuje funkci rozostření objektivu, která umožňuje nastavit rozostření pozadí snímaného objektu.

Hlasové ovládání Aplikace neumožňuje ovládání hlasem.

Úpravy obrazu v reálném čase Žádné úpravy obrazu v reálném čase nejsou dostupné.

Vysoký digitální zoom Nastavení digitálního zoom je omezené, jeho hodnota není v aplikaci uvedena. Pouhým pozorováním bylo zjištěno, že maximální zoom byl u použitého zařízení přibližně 4×.

2.2. Candy Camera - Selfie Camera verze 1.32

Aplikace *Candy Camera*[3] je navržena pro pořizování fotografií s možností aplikací množství obrazových efektů. Uživatelské rozhraní umožňuje zobrazení pouze na výšku a ovládací prvky jsou umístěny u horní a dolní hrany obrazovky.

Základní funkce Primárním účelem je pořizování fotografií a jejich uložení do JPEG. V aplikaci není možné nastavit výsledné rozlišení, v případě použitého zařízení měly výsledné fotografie rozměr 1080×1440. Je možné použití různých obrazových filtrů, deformací a mozaiek. Každou pořízenou fotografii je možné přímo v aplikaci upravit použitím nejrůznějších nástrojů od oříznutí, rotaci, nastavení barev až po přidání rámečků a rozostření. Aplikace neumožňuje nahrávání videa.

Hlasové ovládání Aplikace neumožňuje ovládání hlasem.

Úpravy obrazu v reálném čase Během pořizování fotografií je možné nastavit obrazové filtry a deformace, které uživatel vidí aplikované na živém obrazu z kamery.

Vysoký digitální zoom Digitální zoom je možné nastavit až na maximální hodnotu 10.9×. Koeficient zvětšení je zobrazen pouze během jeho nastavování. Výsledné fotografie jsou pořízeny s digitálním zoom, které je aplikováno na živém náhledu.

2.3. Camera ZOOM FX verze 5.1.0

Camera ZOOM FX[4] je aplikace pro pořizování fotografií za současného přidání různých textur do obrazu. Uživatelské rozhraní zobrazuje ovládací prvky u horní a dolní hrany obrazovky při zobrazení na výšku. Při otočení zařízení na šířku zareaguje uživatelské rozhraní rotací všech ikon o 90 °.

Základní funkce Funkcí aplikace je pořizování fotografií. To je možné v několika režimech: pořízení snímku při nízkých otřesech zařízení, časovač, hlasová aktivace, sekvence fotografií, koláž a časosběrné fotografie. Aplikace neumožňuje nahrávání videa.

Hlasové ovládání Aplikace umožňuje použití hlasového vstupu pro pořízení fotografie. Nepoužívají se příkazy, je pouze měřena hladina zvuku z mikrofónu, kdy při překročení určité meze, je pořízena fotografie.

Úpravy obrazu v reálném čase Možné je pouze překrytí živého náhledu texturami. Aplikace filtrů a jiné zpracování obrazu z kamery není k dispozici.

Vysoký digitální zoom Nastavení digitálního zoom se v aplikaci provádí pomocí jezdce, který obsahuje hodnoty od 1 do 100. Ty však nepřestavují skutečný koeficient zvětšení. Pozorováním bylo zjištěno, že maximální zoom je přibližně 4×.

2.4. Camera MX verze 2.3.4

Aplikace *Camera MX*[5] umožňuje pořizování fotografií s efekty a základní nahrávání videa. U horní a dolní hrany obrazovky se nachází všechny ovládací prvky, které reagují rotací při zobrazení na šířku.

Základní funkce Pořizování fotografií je možné v kombinaci s použitím obrazových filtrů, deformací a překrytím texturou. Aplikace umožňuje záznam videa v rozlišení 1920×1080 bez jakýchkoli efektů. Nahrané video je možné přímo v aplikaci od začátku i od konce oříznout na požadovanou délku.

Hlasové ovládání Aplikace neumožňuje ovládání hlasem.

Úpravy obrazu v reálném čase Na živý náhled z kamery je možné aplikovat různé barevné filtry, deformace a rozostření. Dále je možné překrytí texturou. Na výslednou fotografii budou aplikovány všechny použité efekty, na video však ne.

Vysoký digitální zoom Je možné nastavit maximální zoom 4× tažením dvou prstů od sebe na obrazovce.

2.5. Paper Camera verze 4.0.2

Paper Camera[6] je aplikace pro pořízení fotografií a video záznamu s aplikovanými obrazovými efekty. Grafické uživatelské rozhraní připomíná list zmačkaného papíru, ovládací prvky mají vypadat jako nakreslené pastelkou. Dvě třetiny plochy obrazovky zabírá živý náhled z kamery.

Základní funkce Aplikace fotografuje snímky s aplikovaným barevným efektem, není možné zvolit výchozí rozlišení. Na použitém zařízení byly fotografie pořízeny o velikosti 3264×2448. Nahrávání videa je možné s aplikovaným efektem v rozlišení 640×480 při 25 snímcích za vteřinu.

Hlasové ovládání Aplikace neumožňuje ovládání hlasem.

Úpravy obrazu v reálném čase Je možné použít několik obrazových filtrů pro úpravu živého náhledu z kamery v reálném čase. Každý filtr je možné nastavit několika parametry udávající barevné nastavení, použitou detekci hran a další vlastnosti.

Vysoký digitální zoom Aplikace neumožňuje nastavení digitálního zoom.

2.6. A Better Camera verze 3.19

Mobilní aplikace *A Better Camera*[7] je určena pro pořizování fotografií. Grafické uživatelské rozhraní obsahuje ovládací prvky umístěné u horní a dolní hrany. Dole uprostřed dominuje tlačítko spouště. Všechny ovládací prvky reagují na otočení na šířku rotací.

Základní funkce Pořizování fotografií je možné v několika režimech popisujících zaznamenávanou scénu. Nastavení velikosti snímku není v aplikaci možné, na použitém zařízení byly fotografie pořízeny o velikosti 3264×2448. Nahrávání videa je možné v rozlišení 1920×1080 bez obrazových efektů.

Hlasové ovládání Aplikace neumožňuje ovládání hlasem.

Úpravy obrazu v reálném čase Aplikace umožňuje odstranění pohybujícího se předmětu z fotografie pozorované scény. Je sejmuto několik snímků s jejíchž využitím je pohybující se objekt vymazán.

Vysoký digitální zoom Nastavení digitálního zoom je možné, není zobrazen koeficient zvětšení. Pozorováním byl zjištěn maximální zoom u použitého zařízení přibližně 4×.

2.7. Zoom Camera verze 6.3

Aplikace *Zoom Camera*[8] pořizuje fotografie a videozáznamy. Grafické uživatelské rozhraní obsahuje na celé ploše obraz z kamery a velmi malá tlačítka po obvodu obrazovky. Ty nereagují na stisk vždy správně a nedovolují tak ovládat aplikaci pohodlně.

Základní funkce Je možné pořizovat fotografie o různých rozlišeních, kde u použitého zařízení bylo nejvyšší možné 3264×2448. Video je možné nahrávat v několika rozlišeních, kde nejvyšší možné je 1920×1080.

Hlasové ovládání Aplikace neumožňuje ovládání hlasem.

Úpravy obrazu v reálném čase Je možné nastavit základní barevné obrazové filtry a použít překrytí texturou rámečku. Efekty jsou zobrazeny v živém náhledu a na pořízené fotografii. Video je zaznamenáno bez efektů.

Vysoký digitální zoom Aplikace umožňuje až 10× digitální zoom. Na použitém zařízení však při nastavování zoom docházelo k třesení obrazu.

2.8. Shrnutí

Ze všech uvedených aplikací nebyla žádná, která by umožňovala pokročilé nahrávání videa a jeho upravování v reálném čase v dostatečné kvalitě. Pouze aplikace *Paper Camera* umožňovala použití barevných filtrů při záznamu videa, výsledek však byl v nízkém rozlišení a špatné kvalitě. Vytvořením aplikace, která umožní pokročilou úpravu videa ve vysokém rozlišení už při jeho nahrávání, vznikne unikátní nástroj, který nemá na trhu alternativu.

Hlasové ovládání aplikace pro záznam videa a fotografování se vyskytlo ve výše uvedených případech pouze jednou v případě *Camera ZOOM FX*. Zde se však jedná pouze o primitivní způsob ovládání zvýšením hluku. Aplikace tohoto typu ovládaná hlasovými povely nalezena nebyla a i v tomto případě se tak jedná o řešení, které je unikátní.

3. Aplikace Zoomera 2012

Zoomera 2012 je aplikace pro mobilní operační systém Google Android, která slouží jako digitální lupa a fotoaparát, který využívá vestavěné kamery mobilního zařízení. Vychází z práce [9]. Zoomera 2012 je v této práci použita jako základ pro implementaci nahrávacího modulu a modulu pro ovládání hlasem.

3.1. Použité technologie

Při vývoji aplikace Zoomera 2012 byly použity standardní vývojové knihovny Android SDK spolu s grafickou knihovnou OpenGL ES.

3.1.1. Android SDK

Balík standardních knihoven určených pro vývoj aplikací pro systém Android. Obsahuje všechny nástroje a třídy pro vývoj aplikací, tvorbu uživatelského rozhraní, ovládání systému a hardware.

3.1.2. OpenGL ES

Grafická knihovna OpenGL ES slouží pro renderování 2D a 3D grafiky. Jedná se o podmnožinu grafického systému OpenGL, která je určena pro vestavěné systémy. V aplikaci Zoomera 2012 byla použita knihovna OpenGL ES verze 2.0, která umožňuje přímé programování grafického hardware pomocí shader programů. [10]

Obrázek 2. Grafické uživatelské rozhraní aplikace Zoomera 2012

3.2. Fuknce

Aplikace Zoomera 2012 disponuje grafickým uživatelským rozhraním (viz Obrázek 2 a 3), které umožňuje ovládat několik funkcí, které jsou uvedeny níže.

3.2.1. Digitální zoom

Obraz z kamery fotoaparátu je zpracováván na grafickém čipu zařízení pomocí OpenGL ES. Díky tomu je možné upravovat obraz nezávisle na hardwarových možnostech samotné kamery. To se využívá k aplikaci digitálního zoom. Ten je v aplikaci nastavován v rozmezí 1 až 20 násobné zvětšení. Zvetšení je implementováno jako výřez z textury, ve které se nachází živý obraz z kamery. O vykreslení výřezu se stará knihovna OpenGL, která při samotném vykreslení sama aplikuje potřebné interpolace výřezu obrazu, který je tak vykreslen v rozlišení displeje zařízení. Zoom je možné nastavit pomocí nabídky pod tlačítkem s ikonou lupy v pravém horním rohu GUI (viz Obrázek 2).

3.2.2. Ostření optiky fotoaparátu

Pokud optika zařízení, na kterém je spuštěna aplikace Zoomera 2012 disponuje autofocusem, umožňuje aplikace jeho spuštění. Automatické ostření je možné provádět ve dvou módech. Prvním je provedení jednoho pokusu o zaostření, kdy uživatel stiskne příslušný ovládací prvek, který spustí systémové funkce, které se pokusí nastavit optiku kamery tak, aby byl obraz ostrý. Princip a implementace těchto systémových algoritmů je specifický pro každého výrobce a zařízení. Druhou možností ostření je kontinuální ostření, kdy se stiskem tlačítka AUTO (viz Obrázek 3), nacházející se vedle tlačítka pro ostření, spustí režim nepřetržitého zaostřování. Vždy, když se obraz rozostří, není vyžadován uživatelský vstup a spustí se algoritmus ostření.

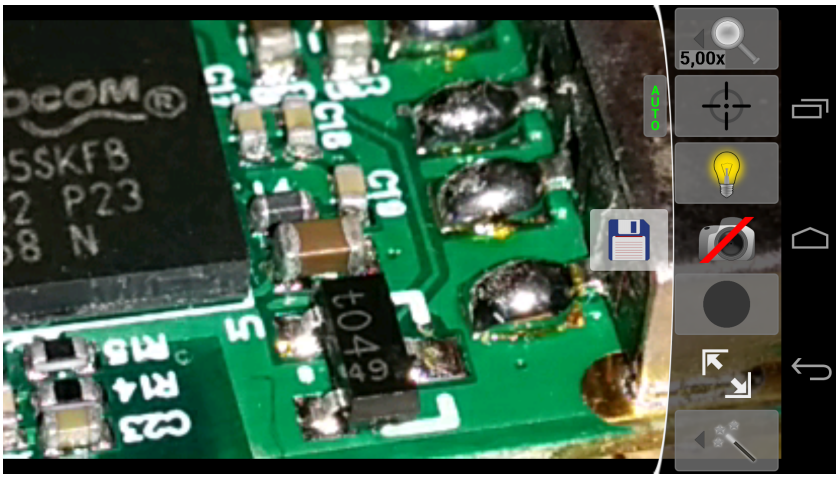
Obrázek 3. Grafické uživatelské rozhraní aplikace Zoomera 2012 s nastaveným digitálním zoom na 5x, zobrazeným panelem pro nastavení zoom, rozsvícenou LED, povoleným kontinuálním ostřením a zobrazenou miniaturou.

3.2.3. Přisvětlení

Disponuje-li kamera zařízení LED přisvětlovací diodou, je možné ji v aplikaci ovládat. O její rozsvícení a zhasínání se stará ovládací tlačítko s ikonou žárovky (viz Obrázek 3).

3.2.4. Fotografování

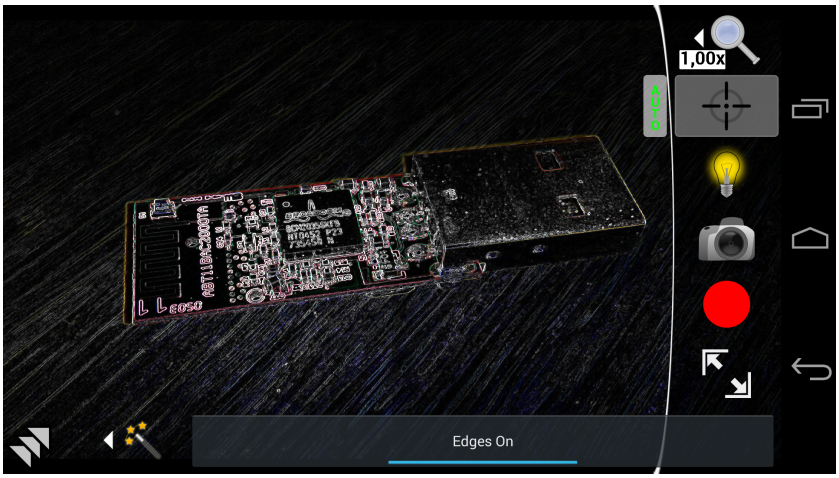
Aplikace umožňuje pořídit fotografii pozorované scény. Pro pořízení snímku se používá systémových volání, jejichž výsledkem je obraz v JPEG formátu. Pokud je v okamžiku stisku tlačítka pořízení fotografie (ikona fotoaparátu, viz Obrázek 3) aplikován digitální zoom, je výsledná fotografie také oříznuta tak, aby odpovídala nastavení živého náhledu před fotografováním (viz Obrázek 4).



Obrázek 4. Grafické uživatelské rozhraní aplikace Zoomera 2012 po vyfotografování scény s nastaveným digitálním zoom na 5x, rozsvícenou LED, povoleným kontinuálním ostřením.

3.2.5. Nahrávání videa

Grafické uživatelské rozhraní aplikace Zoomera 2012 obsahuje tlačítko pro ovládání nahrávání videa (ikona červeného kruhu viz Obrázek 5) funkce v ní však není implementována.



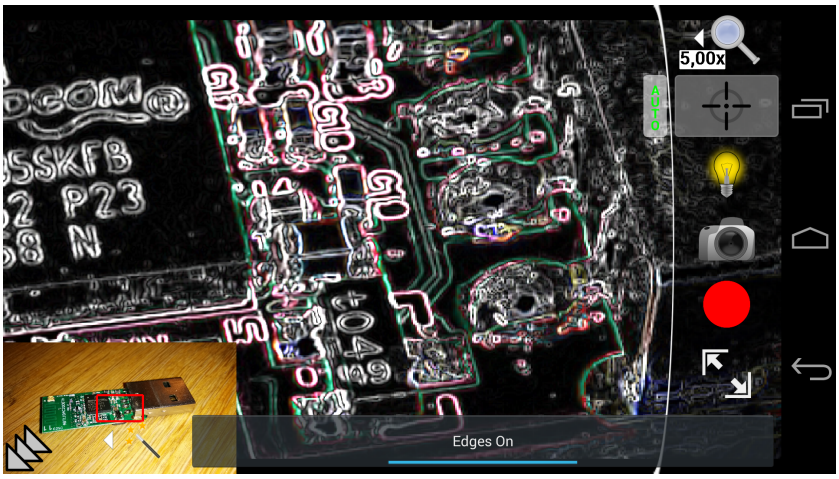
Obrázek 5. Grafické uživatelské rozhraní aplikace Zoomera 2012 se zobrazeným panelem efektů, rozsvícenou LED, povoleným kontinuálním ostřením a aplikovaným obrazovým filtrem na detekci hran.

3.2.6. Miniatura

Při velkých hodnotách nastaveného digitálního zoom začíná být pro uživatele obtížné orientovat se v pozorované scéně. K snížení dopadu tohoto faktu slouží funkce miniatura, která se spouští tlačítkem v levém dolním rohu GUI. Tato funkce zobrazí v levém dolním rohu GUI miniaturu živého náhledu pozorované scény bez aplikovaného digitálního zoom. Obsah červeného obdélníku na této miniatuře ukazuje, jaká část živého náhledu je vykreslována na celou obrazovku. Díky tomu, že je miniatura bez digitálního zoom, umožňuje uživateli rychlejší orientaci ve scéně a snadněji tak může identifikovat přiblížený fragment obrazu (viz Obrázek 3 a 6).

3.2.7. Efekty obrazu

Zpracovávání obrazu z kamery pomocí grafické knihovny OpenGL ES umožňuje upravovat obraz a aplikovat na něj efekty v reálném čase. Zapnutí těchto efektů umožňuje nabídka pod tlačítkem v pravém dolním rohu GUI (tlačítko se symbolem kouzelné hůlky, viz Obrázek 5). Nabídka umožňuje aplikovat na obraz filtr detekce hran. Obraz je zpracováván v reálném čase pomocí shader programů běžících na grafickém čipu (viz Obrázek 5 a 6). Tento konkrétní použitý shader program má implementovanou konvoluci obrazu. Používá 3×3 Sobel filtr pro detekci hran.



Obrázek 6. Grafické uživatelské rozhraní aplikace Zoomera 2012 s nastaveným digitálním zoom na 5x, zobrazeným panelem efektů, rozsvícenou LED, povoleným kontinuálním ostřením, zobrazenou miniaturou a aplikovaným obrazovým filtrem na detekci hran.

4. Použité technologie, nástroje a zařízení

Pro vývoj nové verze aplikace Zoomera, který je předmětem této práce, byly použity následující technologie, nástroje a zařízení.

4.1. Technologie

Technologie určené pro vývoj software použité v této práci jsou uvedeny níže. Všechny jsou standardně dostupné pro vývoj mobilních aplikací na platformu Android.

4.1.1. Android SDK

Pro vývoj na systém Android je určen vývojářský balík Android SDK. Obsahuje řadu nástrojů určených pro vývoj aplikací, obraz systému Android, dokumentaci a potřebná rozhraní a knihovny. Výchozí programovací jazyk, který je pro vývoj používán, je Java.

4.1.2. Android NDK

Vývoj na platformě Android umožňuje použití nativních metod, které jsou napsané v programovacím jazyce C/C++. Android NDK obsahuje nástroje, dokumentaci a potřebná rozhraní a knihovny, které jsou nezbytné pro možnost použití nativního vývoje. Pro propojení Javy a nativního kódu se používá rozhraní JNI.

4.1.2.1. JNI

Java Native Interface (JNI) je rozhraní programovacího jazyka Java, který umožňuje propojení Java Virtual Machine s programy napsanými v nativních programovacích jazycích C a C++.

4.1.3. OpenGL ES

OpenGL ES je grafická knihovna, která umožňuje řízení vykreslování obrazu a definuje rozhraní a volání pro programování grafického hardware pomocí shader programů. Způsob ovládání knihovny je možné popsat stavovým automatem, je tedy nutné dodržovat správné pořadí volání příkazů při jejím nastavování před spuštěním samotného vykreslování.

4.1.3.1. Shader jazyk

Shader jazyk je programovací jazyk podobný jazyku C. Slouží pro programování grafického hardware pomocí jednoduchých shader programů. Existují dva typy shaderů, ze kterých se skládá shader program: vertex a fragment shader. Vertex shader pracuje s vykreslovaným obrazem jako s celkem a nastavuje jeho geometrii. Fragment shader slouží k řízení vykreslování jednoho konkrétního bodu obrazu.

4.2. Nástroje

Během tvorby této práce byly použity následující vývojářské nástroje.

4.2.1. Eclipse

Eclipse je vývojářské prostředí, které umožňuje komplexní správu vývoje od psaní zdrojových kódů, přes jejich kompilaci, spouštění, debugování a verzování. Během celého vývoje byla používán software Eclipse ve verzi 3.8. Pro verzování vyvíjeného kódu byl použit systém Git.

4.3. Zařízení

4.3.1. LG Nexus 5

Pro vývoj a testování bylo použito mobilní zařízení LG Nexus 5 [11]. Vývoj byl cílen na funkčnost na tomto zařízení, není vyloučeno že se na jiných zařízeních může aplikace chovat odlišně.

Velikost displeje: 5"
Rozlišení displeje: 1920×1080 bodů"
Hmotnost: 129 g
Vnitřní paměť: 16 GB
CPU: quad core 2.3 GHz
Verze systému Android: 4.4.2
Verze základního pásma: M8974A-1.0.25.0.23
Verze jádra: 3.4.0-gadb2201
Číslo sestavení: KOT49H

Obrázek 7. Mobilní telefon LG Nexus 5 [11]

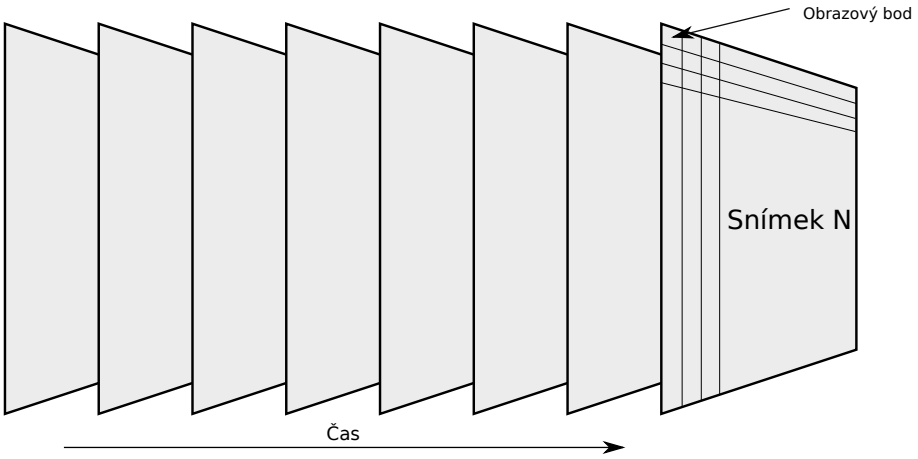
Rozměry (š × v × h): 137,8 mm × 69,1 mm × 8,5 mm
Fotoaparát (hlavní/vzadu): 8 MPx, autofocus, LED/1,3 MPx
Bezdrátové připojení: Wi-Fi 802.11 a/b/g/n/ac, Bluetooth 4.0, NFC

5. Nahrávání videa

Pro získání plné kontroly nad nahrávaným videem je nutné zpracovávat obraz z kamery na GPU zařízení. To umožní efektivní přístup k úpravě obrazu, který musí být následně dostatečně rychle zkomprimován, aby nedošlo ke ztrátě následujících snímků. Tato část práce pokrývá všechny podstatné oblasti, které je nutné pro dosažení výše zmíněného cíle použít. Obsahuje základy ukládání a manipulace s digitálním videem, především kompresním formátem H.264/MPEG-4 AVC a multimediálním kontejnerem MP4, které byly při implementaci aplikace použity. Dále se zabývá zpracováním video snímků a možnostmi zaznamenávání videa v systému Android spolu s nahráváním videa v nekonečné smyčce. Obsažen je popis způsobu implementace jednotlivých technologií v aplikaci Zoomera.

5.1. Digitální video

Video je možné definovat jako sekvenci snímků, které jsou postupně zobrazovány na výstupním zařízení a člověk je pozoruje jako pohybující se obrazy. Jedná se o digitální medium, které má definovány principy záznamu, editace a přehrávání. Video je spojováno s televizí, počítači a internetovým přenosem. Velmi často je video stopa šířena s jednou či více audio stopami a případně i dalšími multimediálními daty (titulky, časové značky, obrazové materiály a další).



Obrázek 8. Zjednodušený princip digitálního videa.

Digitální video je složeno z obdélníkových snímků, které se mění v pravidelném intervalu (každý má vlastní časovou značku) a umožňují tak zaznamenání změny scény či pohybu. Každý snímek je samostatný digitální obraz, který je tvořen obrazovými body, kde každý obrazový bod v případě barevného videa nese jasovou hodnotu každé barevné složky. Takto definované video nese velké množství informace a je tak velmi datově náročné. Z toho důvodu je časté provádění ztrátové komprese videa. Ta umožní snížit datové nároky, spolu s nimi však dochází k ztrátě části obrazové informace.

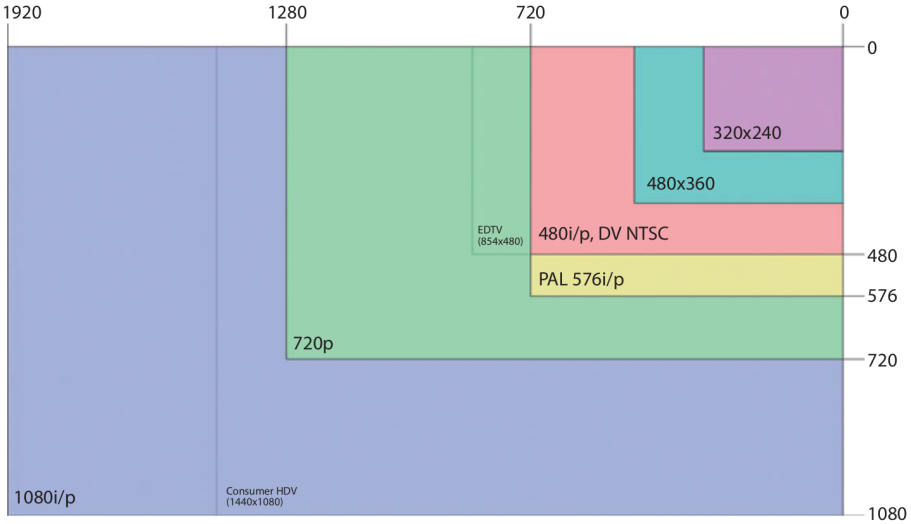
Parametry videa Digitální video je proud dat, který má své specifické parametry, které ovlivňují kvalitu obrazové informace, množství zachyceného detailu vzhledem k obrazovým bodům, plynulost zachycených pohybů a věrnost barev.

5.1.1. Snímková frekvence

Obrazové snímky zaznamenávají vždy jeden okamžik scény, která byla snímána. Celé video je poskládáno z velkého množství takových snímků. Lidský mozek má funkci, která umožňuje člověku vnímat plynulý pohyb při sledování videa, i když se skládá z jednotlivých nehybných snímků. Věrnost tohoto vjemu závisí na tom, kolik snímků je v určitém časovém intervalu zobrazeno. Tato hodnota je udávána v zobrazených snímcích za sekundu, značí se zkratkou *fps* (frames per second). Nejčastěji se setkáváme se snímkovací frekvencí 25 fps, resp. 30 fps. S vyšší snímkovou frekvencí se zvyšuje i objem video dat, protože je pro určitý časový úsek uložit větší počet snímků. [12, 13]

5.1.2. Rozlišení

Rozlišení video snímku udává kolik obrazových bodů se nachází v jednom řádku, respektive v jednom sloupci. Jinak vyjádřeno, udává šířku a výšku videa v obrazových bodech. Nejčastěji se značí *šířka*×*výška* v obrazových bodech (viz Obrázek 9), případně jen *výška* spolu se symbolem značící použití prokládání (viz 5.1.3). Šířka i výška snímku by měla být dělitelná číslem 16, je to požadováno pro použití kompresních algoritmů, které rozdělují každý snímek na makrobloky o velikosti 16×16 obrazových bodů (viz Kapitola 5.2.8). [13, 14]

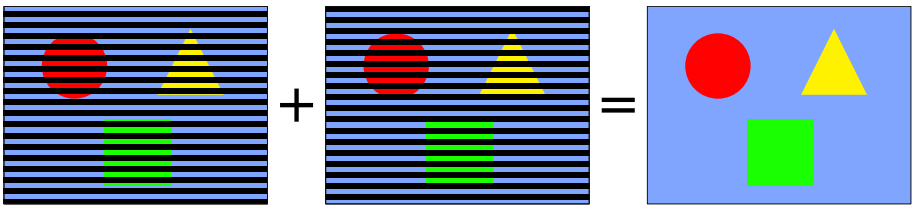


Obrázek 9. Nejčastější rozlišení digitálního videa. Reference: [13]

5.1.3. Progresivní a prokládané video

Jednotlivá videa se mohou lišit tím, jakým způsobem je zobrazován snímek při přehrávání. Existují dva typy: progresivní a prokládané.

Progresivní video je první případ, kdy se při každém vykreslení zobrazí celý snímek. Říká se mu také neprokládané video. [14]

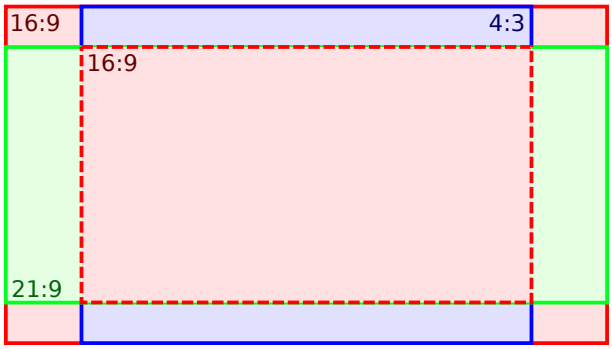


Obrázek 10. Princip půlsnímků u prokládaného videa. Snímek s lichými řádky spolu se snímkem se sudými řádky dávají dohromady výsledný obraz.

Prokládané video je druhým případem. Tento způsob funguje tak, že je vždy vykreslena jen polovina řádků konkrétního snímku (tzv. půlsnímek). Zobrazení funguje tak, že v *i*-tém snímku se vykreslují pouze liché řádky snímku a následně v *i+1* snímku se vykreslí pouze sudé řádky snímku (viz Obrázek 10). Výhody tohoto způsobu spočívají v tom, že při stejném objemu dat můžeme zdvojnásobit snímkovou frekvenci videa.
Máme progresivní video s rozlišením 640×480 a snímkovou frekvencí 25 fps. V tomto případě zobrazujeme 640×480 bodů jednou za 1/25 s. V případě, že by toto video bylo prokládané, měl by každý snímek rozlišení 640×240 obrazových bodů a vykreslovali bychom v intervalu 1/50 s.
Nevýhodou prokládaného videa je, že zobrazovací zařízení v jednom okamžiku zobrazuje na lichých řádcích snímek z času t_i a zároveň na sudých řádcích snímek z času t_{i+1}. Při velkých rozdílech mezi dvěma sousedními řádky ze snímku *i* a snímku *i+1* může docházet k nežádoucím obrazovým defektům. Filtrování dolní propustí může tento vedlejší efekt prokládání odstranit.
Můžeme se setkat z označením, které udává rozlišení, a také zda je video progresivní nebo prokládané. Značí se *počet-řásků*p pro progresivní video a *počet-řádků*i pro prokládané video. Často používané označení tak jsou 1080p a 1080i pro prokládané, resp. neprokládané video o rozlišení 1920×1080, případně též 720p a 720i. [13, 12, 14]

5.1.4. Poměr stran

Poměr stran videa je definován jako poměr šířky a výšky video snímku. Ve smyslu tvorby video záznamu se nejčastěji se setkáváme s poměry stran 4:3, 16:9 a 21:9 (viz Obrázek 11). Pro správné zobrazení videa je potřeba zohlednit poměr stran zobrazovacího zařízení, poměr stran video snímku tak, jak je uložen (*výška*×*šířka* obrazových bodů), a poměr, který je nastaven ve video kontejneru (viz Kapitola 5.2.9.1). [13]



Obrázek 11. Vybrané poměry stran obrazu používané v digitálních videích.

5.1.5. Datový tok

Datový tok videa udává velikost dat, které obsazuje video záznam o délce jednotky času. Datový tok videa se udává v megabitech za sekundu a má jednotku *Mbps*. Pro video soubor se udává průměrný datový tok přes celou délku záznamu. Pokud se jedná o aktuální datový tok přehrávaného videa, udává se průměr datového toku přes malý časový úsek záznamu v okolí právě přehrávané pozice.

5.1.6. Prostor barev

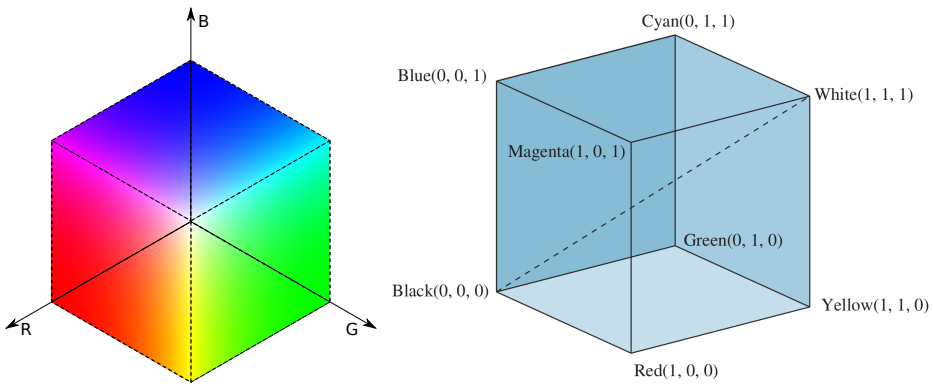
Prostor barev je matematické vyjádření umožňující popsat barvu sadou čísel. Níže jsou uvedeny nejčastější prostory barev, zejména ty, které se vyskytují v procesu tvorby videí. Určením prostoru barev pro snímek můžeme také určit, jak velká bude informace, kterou ponese jeden obrazový bod. Čím větší informace bude, tím větší také bude paleta barev, kterou můžeme použít. S tímto však také je přímo ovlivněn objem dat, které video zaplní.

5.1.6.1. RGB

Barevný prostor používaný zejména v počítačové grafice. Prostor je definován trojicí čísel, kde každá zastupuje jednu z barevných složek: R - red/červená, G - green/zelená, B - blue/modrá. Vyjádření základních barev je v Tabulce 1 a na Obrázku 12). V případě, že chceme vyjádřit také průhlednost, přidává se čtvrtý barevný kanál A a vzniká barevný prostor RGBA. [15, 12, 14]

	červená	**zelená**	**modrá**	**černá**	**bílá**
R	1	0	0	0	1
G	0	1	0	0	1
B	0	0	1	0	1

Tabulka 1. Vyjádření základních barev v RGB prostoru.



Obrázek 12. RGB barevná krychle. Na obrázku vpravo je přerušovanou linií naznačena hlavní diagonála, která představuje šedivou barvu. Reference: [16, 15]

5.1.6.2. CMY

Barevný prostor používaný převážně v oblasti tisku, C - cyan/azurová, M - magenta/purpurová, Y - yellow/žlutá. Na rozdíl od barevného prostoru RGB (viz Kapitola

5.1.6.1) nedostaneme sloučením všech barevných složek bílou barvu, ale barvu černou. Tiskárny používají upravený model CMYK, kde je přidána složka černé barvy (K - black). Převod CMY barvy do barevného profilu RGB se provede následovně [15]:

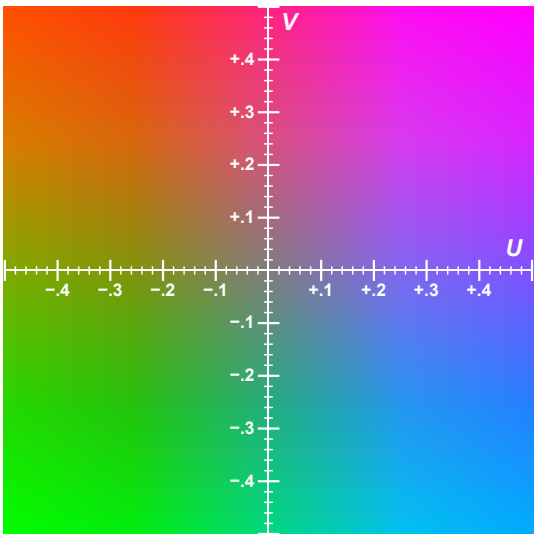
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 - C \\ 1 - M \\ 1 - Y \end{bmatrix}$$

5.1.6.3. YUV

Barevný prostor, který se skládá ze tří složek: Y - jas a barevných složek U a V. Převážně používaný v televizní technice. Jasová složka sama o sobě nese informaci o černobílém obrázku, složky U a V udávají barvu podle poloze na UV rovině (viz Obrázek 13). Rovnice pro převod mezi RGB a YUV jsou následující [14]:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & 0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.140 \\ 1 & -0.395 & -0.581 \\ 1 & 2.032 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$



Obrázek 13. UV rovina z barevného prostoru YUV pro Y = 0,5. Reference: [17]

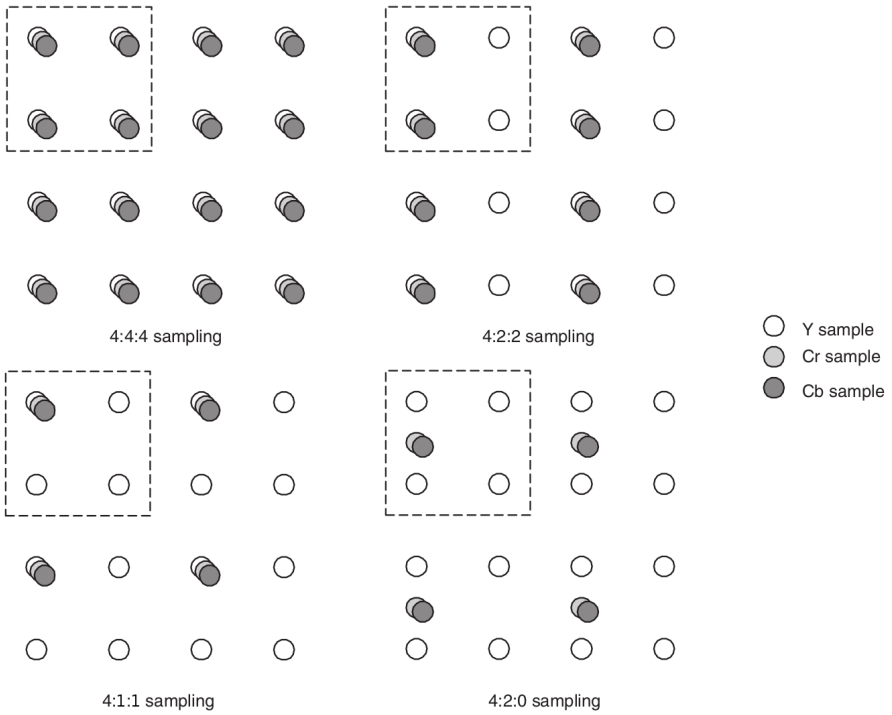
5.1.6.4. YCbCr

Barevný prostor YCbCr je podobný prostoru YUV (viz Kapitola 5.1.6.3). Složka Y představuje jas a Cb a Cr jsou barevné složky. Rovnice pro převod mezi RGB a YCbCr jsou následující [12, 14]:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.391 & -0.813 \\ 1.164 & 2.018 & 0 \end{bmatrix} \begin{bmatrix} Y - 16 \\ Cb - 128 \\ Cr - 128 \end{bmatrix}$$

YCbCr vzorkovací formáty Formát YCbCr nabízí různé možnosti vzorkování barevných složek. Jsou definovány počtem obrazových bodů, které přísluší dané složce (Y:Cb:Cr). Existují následující možnosti vzorkování: 4:4:4, 4:2:2, 4:1:1 a 4:2:0 (viz Obrázek 14). V případě vzorkování 4:2:2 tak platí, že na každý pixel připadá jedna složka jasu Y a na každé dva pixely složky Cb a Cr. Chromatické komponenty tak mají poloviční rozlišení než komponenta jasu. [12, 14]



Obrázek 14. Přehled YCbCr 4:4:4, 4:2:2, 4:1:1 a 4:2:0 vzorkování. Reference: [12]

Způsob uložení do souboru Existují dvě základní rozdělení ukládání obrazových dat v barevném prostoru YUV. Prvním je planární a druhý je semi-planární způsob. Liší se způsobem uložení jednotlivých kanálů Y, U a V. [18]

Planární způsob je založen na třech rovinách, kde každá rovina obsahuje jeden barevný kanál. Jedna plocha obsahuje jasovou složku Y a další dvě plochy obsahují barevné složky U a V. [18]

Semi-Planární způsob je založen na dvou rovinách, kde jedna plocha obsahuje jasovou složku Y a druhá plocha obsahuje obě barevné složky U a V, které jsou na ploše rovnoměrně rozprostřeny. [18]

5.2. Komprese videa

Video je datově velmi náročný multimediální element. Aby mělo pro diváka dostatečný přínos, musí mít vysokou snímkovou frekvenci, velké množství obrazových bodů na snímek a dostatečnou barevnou hloubku. Všechny tyto faktory ovlivňují výslednou velikost videa a bez aplikace komprese dat by bylo pro svůj velký objem velmi obtížné video ukládat a distribuovat. Z tohoto důvodu bylo vytvořeno mnoho kompresních algoritmů, které umožní objem dat zmenšit za případnou cenu ztráty některých detailů. Využívají metod k odstranění redundance dat ve video snímcích, tedy není nutné si pamatovat všechna data, ale jen část a zbytek je možné z této části zrekonstruovat.

5.2.1. Bezztrátová komprese

Název této metody napovídá, že se jedná o kompresi, při které nedojde ke ztrátě komprimované informace a je možné ji celou znovu obnovit. Nevýhodou tohoto způsobu komprese je, že není efektivní, tedy místo, které je ušetřeno není velké vzhledem k náročnosti zpracování. Z tohoto důvodu se tato metoda používá jen v malém množství případů. [12, 13]

5.2.2. Ztrátová komprese

Oproti bezztrátové kompresi, během ztrátové komprese dochází k nenávratné ztrátě informace. Když data dekódujeme, dostaneme video, které není totožné s původním záznamem, jedná se o jeho aproximaci. Čím vyšší kompresní poměr zvolíme, tím méně dat výsledné video zabere, ale také ztratíme více informací a dekódované video bude horší aproximací původního záznamu. Cílem je nalézt takový kompresní poměr, aby video nepotřebovalo velký datový prostor, ale zároveň bylo pro diváka přínosem. [12]

5.2.3. Prostorová redundance

Zkoumá video snímky jednotlivě, nezávisle na ostatních. Předpokladem je, že mezi jednotlivými obrazovými body je vysoká korelace, tedy, že jednotlivé obrazové body si jsou podobné. Nejčastěji jsou si sousední obrazové body podobné na velkých barevných plochách, odlišné sousední obrazové body nacházíme nejčastěji na hranách. [13]

5.2.4. Časová redundance

Zkoumá video snímky scény, které byly zachyceny v krátkém časovém intervalu, a porovnává rozdíly mezi nimi. Předpokládá se, že mezi sousedními snímky je velká korelace, tedy že rozdíly sousedních snímků budou velmi malé. Platí, že čím vyšší je snímkovací frekvence videa, tím je pravděpodobnost, že si jsou sousední snímky podobné, větší. [13]

5.2.5. Intraframe komprese

Metoda spočívá v kompresi jednotlivých snímků nezávisle na ostatních. Využívá se pouze prostorové redundance dat a pomocí diskrétní kosinové transformace nebo wavelet transformace jsou obrazová data komprimována. [13, 12]

5.2.6. Interframe komprese

Metoda využívá časovou redundanci dat a snaží se najít podobnosti mezi jednotlivými sousedními snímky. Existuje jeden referenční snímek (viz Kapitola 5.2.7), který je uložen celý, a vzhledem k němu jsou následující snímky porovnávány a jsou z nich ukládány jen rozdíly od referenčního snímku. Na tyto rozdíly je aplikována komprese pomocí DCT nebo wavelet transformace. [13]

5.2.7. Typy snímků

V případě použití interframe komprese se setkáváme s několika typy snímků. [13]

5.2.7.1. I-frame

Jinak také nazývaný intra frame nebo key frame, je takový snímek, který je uložen celý, nezávisle na sousedních snímcích. Všechny ostatní typy snímků jsou tvořeny v závislosti na některém z I-snímků. Takový typ snímku je v pravidelných intervalech do videa ukládán, aby v případě poškození jednoho z I-snímků bylo možné alespoň část videa přehrát. [13]

5.2.7.2. P-frame

Predicted snímek využívá nejbližší předchozí I-snímek jako referenční a obsahuje pouze informace o změnách vzhledem k referenčnímu snímku. Pro uložení P-snímku je tak potřeba méně datového prostoru než v případě I-snímku. [13]

5.2.7.3. B-frame

Bidirectional snímek využívá nejbližší dva předchozí I-snímky a pomocí nich predikuje pohyb. Vyžaduje méně datového prostoru než P-snímek. Není ho možné samostatně dekódovat, vyžaduje informace i z dalších několika snímků. [13]

5.2.8. Video kodek

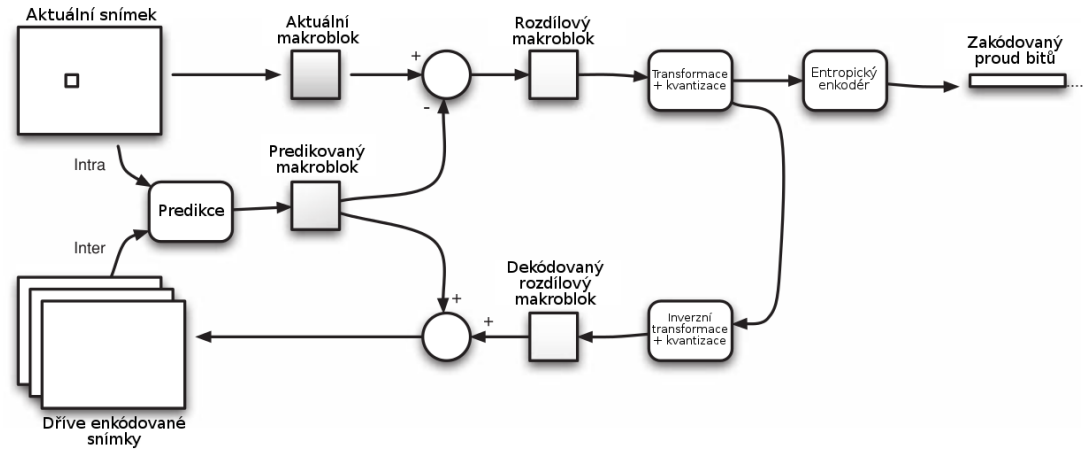
Slovo kodek je odvozeno ze slov enkodér a dekodér [12]. Jedná se o software, který pomocí definovaného algoritmu (enkodér) umožňuje provést kompresi videa a snížit tak jeho paměťové nároky. Tomuto procesu se říká enkódování. Pro přehrání komprimovaného videa je zapotřebí použití stejného kodeku, kterým bylo provedeno enkódování. Použije se dekodér, který převede komprimovaná video data na jednotlivé snímky. Proces enkódování videa bývá zpravidla náročnější než proces dekódování. [13]

Enkódovaný obraz je rozdělen na *makrobloky*, nejčastěji o velikosti 16×16, 8×8 nebo 4×4 obrazových bodů. Tyto *makrobloky* se používají pro nalezení rozdílů mezi sousedními snímky. Ve snímku v čase t_i se vezme konkrétní *makroblok* a vzhledem ke snímku v čase t_{i+1} se uloží rozdíl s odpovídajícím *makroblokem*, případně se hledá takový translační vektor, který odpovídá posunu *makrobloku* v obraze. Výstupem kodeku jsou surová zakódovaná data. [13]

5.2.8.1. H.264/MPEG-4 AVC kodek

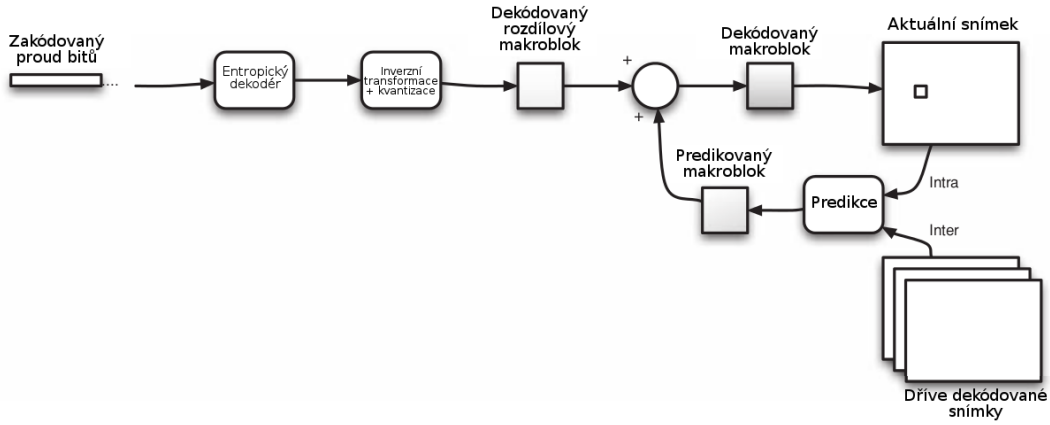
H.264/MPEG-4 AVC je ztrátový video kodek, který je často používán v komerčních aplikacích a přístrojích. Byl publikován v roce 2003 a je definován ISO standardem

[19]. Podporuje snímky ve formátu YCbCr 4:4:4, 4:2:2, 4:2:0 (viz Kapitola 5.1.6.4) a případně i monochromatické 4:0:0.



Obrázek 15. Princip fungování H.264/MPEG-4 AVC enkodéru. Reference: [12]

Kodek při enkódování (viz Obrázek 15) provádí zpracování každého snímku po *makroblocích* o velikosti 16×16 obrazových bodů. Od každého *makrobloku* aktuálně zpracovávaného snímku se odečte makroblok, který je kodekem predikován z předchozích snímků. Vznikne rozdílový *makroblok*, na který je aplikována DCT. Jedná se o proces kvantizace, koeficienty získané z DCT jsou zaokrouhleny a ponechány jen ty s nejvyšší hodnotou. Získáme tak výsledek transformace v nižší dimenzi prostoru. Při zpětném převodu zůstanou zachovány základní artefakty obrazu, ale výsledek bude jen aproximací původního obrazu. Sníží se tak ale datové nároky na jeden *makroblok*. Čím větší kompresní poměr, tím méně přesně se koeficienty DCT ukládají a tím je také výsledný obraz vzdálenější od původního. Kvantizovaný *makroblok* je zakódován některou z forem entropického kódování, aby bylo zaručeno, že bude uložen do nejmenšího možného datového prostoru. Aktuální rozdílový *makroblok* se použije při zpracovávání dalších snímků. [12]



Obrázek 16. Princip fungování H.264/MPEG-4 AVC dekodéru. Reference: [12]

Dekodér pracuje tak (viz Obrázek 16), že přečte zakódovaný *makroblok*, provede inverzní DCT a získá tak aproximovaný rozdílový *makroblok*. Ten se přičte k kodekem

predikovanému *makrobloku* a vznikne tak dekódovaný *makroblok*, který je možné zobrazit. [12]

5.2.9. Struktura video souboru

Výstup z video kodeku je bitový proud, který obsahuje zakódované snímky. Tato data však není možné přehrát, aniž by k nim byly dodány dodatečné meta informace.

5.2.9.1. Multimediální kontejnery

Multimediální kontejnery slouží jako obálka pro jednu nebo více multimediálních stop. Umožňuje archivaci multimédií v podobě souboru, nese základní informace o obsažených médiích a slouží jako zdroj pro přehrávání médií. Usnadňují vyhledávání ve videu a náhodný přístup na konkrétní časovou značku. Synchronizují mezi sebou jednotlivé multimediální stopy. Multimediální kontejnery přináší řadu výše zmíněných výhod v případě, že přehráváme či editujeme video za použití příslušných nástrojů nebo knihoven.

5.2.9.2. MP4 kontejner

MP4 je standardizovaný multimediální kontejner definovaný v [20] v roce 2003 a je založen na standardizované definici multimediálního formátu souboru z [21]. Určený je pro uchovávání video a audio obsahu spolu s doplňkovým obsahem (titulky, časové značky, obrazové materiály a další). Dle [20] je doporučená přípona souboru "*.mp4*". Kontejner umožňuje uložení jedné nebo více video stop (nejčastěji ve formátech H.264/MPEG-4 AVC nebo MPEG-2), dále jednu nebo více audio stop (nejčastěji ve formátu MP3 nebo AAC). Kontejner je objektově orientovaný soubor, který se skládá z prvků zvaných *box* (případně také *atom*). Veškerá data jsou obsažena uvnitř boxů, každý *box* může obsahovat další *boxy*. Vzhledem k multimediální prezentaci jsou nejdůležitější *boxy* typu *ftype*, *mdat* a *moov* (viz Tabulka 2).

Typ boxu	**Popis boxu**
ftype	Obsahuje základní informace o souboru samotném (typ, verze).
mdat	Obsahuje surová multimediální data, případně odkaz na ně.
moov	Uchovává meta informace o multimediálních stopách.

Tabulka 2. Typy základních boxů mp4 kontejneru.

5.3. Zpracování videa z kamery v systému Android

Systém Android umožňuje tři základní způsoby, jak získat a následně zpracovávat video z kamery zařízení v reálném čase. Níže zmíněnými způsoby je možné zpracovávat živý náhled z kamery, řídit způsob kódování, upravovat délku záznamu a případně aplikovat obrazové úpravy a efekty. Následují způsoby získání obrazových dat pro další úpravy: datová kopie snímků, textura v OpenGL ES a funkce *glReadPixels()*.

5.3.1. Kopie snímků

Systémová třída *Camera* [22] umožňuje nastavit callback, který zajistí získání kopií snímků z kamery. Funkce je dostupná po volání metody *setPreviewCallbackWithBuffer(Camera.PreviewCallback cb)* na objektu typu *Camera*. Metoda musí být zavolána po úspěšném spuštění živého náhledu z kamery. Objekt v argumentu metody musí implementovat rozhraní *Camera.PreviewCallback* [23], kde je definována metoda *onPreviewFrame(byte[] data, Camera camera)*, která je volána v případě, že je snímek z kamery dostupný ke zpracování (viz Obrázek 17).

```
1   public class CameraVideoRecorder implements PreviewCallback {
2      // Pocet bufferu pro snimky
3      private final int NUMBER_OF_BUFFERS = 20;
4
5      /**
6       * Nastavi callback pro kopie snimku z kamery. Kdyz je
7       * snimek v kamere dostupny, system zavola metodu onPreviewFrame()
8       * na objektu v parametru metody Camera.setPreviewCallbackWithBuffer().
9       */
10     public void setCameraPreviewCallback() {
11        this.camera.setPreviewCallbackWithBuffer(this);
12     }
13
14     /**
15      * Metoda zavolana kdyz je dostupna kopie snimku z kamery.
16      */
17     @Override
18     public void onPreviewFrame(byte[] data, Camera camera) {
19        // Zpracovani obrazovych dat
20     }
21  }
```

Obrázek 17. Kostra zdrojového kódu, který umožňuje získávání kopie snímků z kamery zařízení.

Data kopie snímku získána v callbacku se nachází v byte poli. Tato pole musí být inicializována v závislosti na rozlišení snímků a použitém barevném formátu (viz Kapitola 5.1.6). Metoda *int getBufferSize(Camera camera)* vypočítá a vrátí minimální potřebnou velikost bufferu při konkrétním nastavení objektu typu *Camera* (viz Obrázek 18). Výchozí barevný formát kopií snímků je NV21, což je obdoba barevného formátu YCbCr 4:2:0 (viz Kapitola 5.1.6.4) [23].

Pro získání kopií snímků je potřeba inicializovat tato pole a předat je pomocí volání metody *addCallbackBuffer()* objektu typu *Camera* (viz Obrázek 18). Ve chvíli, kdy je v kameře dostupný nový snímek, systém nahraje jeho kopii do jednoho z poskytnutých byte polí a předá ho v argumentu volání callback metody. Ve chvíli, kdy uživatelská aplikace zpracuje poskytnuté pole, musí ho předat zpět objektu *Camera*, aby bylo zajištěno kontinuální získávání nových snímků [22].

```
/**
 * Vrati pozadovanou velikost bufferu pro snimek
 * v zavislosti na rozliseni nahledu z kamery.
 */
private int getBufferSize(Camera camera) {
   Parameters parameters = this.camera.getParameters();
   int previewFormat = parameters.getPreviewFormat();
   int bitsPerPixel = ImageFormat.getBitsPerPixel(previewFormat);
   float bytesPerPixel = bitsPerPixel/8.0f;
   Size previewSize = parameters.getPreviewSize();
   return
       (int)Math.ceil((previewSize.width*previewSize.height)*bytesPerPixel);
}

/**
 * Vygeneruje buffery pro snimky.
 */
private void generateBuffers() {
   final int bufferSize = this.getBufferSize(camera);
   for (int i = 0; i < NUMBER_OF_BUFFERS; i++) {
      camera.addCallbackBuffer(new byte[bufferSize]);
   }
}
```

Obrázek 18. Zdrojový kód, který generuje buffery pro získávání kopie snímků z kamery zařízení.

5.3.2. Textura v OpenGL ES

Druhou možností je použití OpenGL ES textury, do které se ukládají snímky z živého náhledu kamery zařízení. Tato metoda byla již použita v [9] pro zpracování obrazu z kamery před zobrazením. Díky metodě je možné při vykreslování obrazu použití shader programů, určených pro programování grafického čipu zařízení. Efektivně se tak dají zpracovávat grafické transformace a aplikovat obrazové filtry.

```
public class CameraPreview extends GLSurfaceView {
   private RenderObject renderObject;
   // ...
   public CameraPreview (ZoomeraActivity activity) {
      // ...
      renderObject = new RenderObject(activity, this);
      this.setRenderer(renderObject);
      // ...
   }

   public void startCameraPreview() {
      // ...
      camera.setPreviewTexture(renderObject.getSurfaceIn());
      camera.startPreview();
   }
   // ...
}
```

Obrázek 19. Zdrojový kód, který demonstruje incializaci *Camera* objektu při použití OpenGL ES textury.

Pro použití této metody se vyžaduje též inicializace OpenGL ES knihovny. Systém Android umožňuje inicializaci a nastavení kontextu provést automaticky při využití třídy GLSurfaceView [24]. Tato třída jako prvek uživatelského rozhraní umožňuje renderování grafiky pomocí OpenGL ES. Samotný průběh vykreslování je řízen objektem, který implementuje rozhraní *GLSurfaceView.Renderer* [25] a nastavuje se voláním metody *setRenderer(Renderer)* (viz Obrázek 19).

```java
public class RenderObject implements Renderer, OnFrameAvailableListener {
   private boolean textureUpdate = false;
   private SurfaceTexture surfaceIn;
   private int surfaceInID;
   // ...
   public void onSurfaceCreated(GL10 gl10, EGLConfig config) {
      // ...
      createGLESTexture();
      // ...
      cameraPreview.startCameraPreview();
   }

   private void createGLESTexture() {
      int[] texArray = new int[1];
      GLES20.glGenTextures(1, texArray, 0);
      surfaceInID = texArray[0];
      GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
      GLES20.glBindTexture(GL_TEXTURE_EXTERNAL_OES, surfaceInID);
      surfaceIn = new SurfaceTexture(surfaceInID);
      surfaceIn.setOnFrameAvailableListener(this);
      GLES20.glBindTexture(GL_TEXTURE_EXTERNAL_OES, 0);
   }

   public void onDrawFrame(GL10 gl10) {
      updateTexture();
      // ... pouzij shadery, vykresli
   }

   private void updateTexture() {
      synchronized(this) {
         if (textureUpdate) {
            surfaceIn.updateTexImage();
            textureUpdate = false;
         }
      }
   }

   public void onFrameAvailable(SurfaceTexture surfaceTexture) {
      synchronized (this) {
         textureUpdate = true;
      }
   }
   // ...
}
```

Obrázek 20. Zdrojový kód, který demonstruje vytvoření, obnovování a vykreslení textury s živým náhledem z kamery.

Objekt *Renderer* řídí životní cyklus a vykreslování pomocí metod *onSurfaceCreated(GL10, EGLConfig)*, *onSurfaceChanged(GL10, int, int)* a *onDrawFrame(GL10)*. Při vytvoření povrchu pro vykreslování je zavolána metoda *onSurfaceCreated()*, kde dojde k vytvoření OpenGL ES textury a následnému spuštění živého náhledu kamery (viz Obrázek 20).

Textura je generována metodou *glGenTextures()*, následně aktivována pomocí *glActiveTexture()* a je ovládána objektem *SurfaceTexture* [26]. Nastavením callbacku *SurfaceTexture.setOnFrameAvailableListener()* v argumentu s objektem implementujícím rozhraní *OnFrameAvailableListener* [27] zajistíme, že systém bude pravidelně signalizovat připravení snímku k uložení do textury. Při každém vykreslení v metodě *onDrawFrame()* dojde vždy k pokusu o obnovení textury a následnému vykreslení na povrch s možnou aplikací úprav obrazu při použití adekvátních shader programů.

5.3.3. Funkce glReadPixels

Pokud uvažujeme situaci, kdy je obraz z kamery vykreslen na libovolném OpenGL ES povrchu, je možné obrazová data z tohoto povrchu uložit do datové struktury *Buffer* [28]. Paměť, do které se uloží kopie obrazových bodů z OpenGL ES povrchu, se alokuje metodou *ByteBuffer.allocateDirect()* (viz Obrázek 21). Vytvoří se tak byte pole o velikosti šířka×výška čtené oblasti v obrazových bodech, násobená číslem 3 (předpokládáme 1 byte na barevný kanál a RGB barevný prostor - viz 5.1.6.1).

Samotné čtení z povrchu se provádí voláním funkce GLES20.glReadPixels() [29, 30]. Při volání se specifikuje oblast povrchu určená ke čtení souřadnicemi levého horního rohu a pravého dolního rohu obdélníku a předpokládá se, že souřadnicový systém povrchu začíná v horním levém rohu. Dále se specifikuje barevný prostor RGB pro uložení obrazových dat *GL_RGB*, datový typ byte uložených dat *GL_UNSIGNED_BYTE* a cílový *Buffer readBuffer*.

Při čtení z OpenGL ES povrchu dochází ke kopírování obrazových dat, která jsou uložená v grafické paměti zařízení, do operační paměti zařízení. Během této operace je zastaveno jakékoli vykreslování knihovnou OpenGL ES a čeká se, dokud nejsou data přečtena a zapsána do požadovaného úseku v paměti.

```
1  private Buffer readBuffer = ByteBuffer.allocateDirect(
2                              3*readAreaWidth*readAreaHeight);
3  GLES20.glReadPixels(0, 0, readAreaWidth, readAreaHeight, GLES20.GL_RGB,
4                      GLES20.GL_UNSIGNED_BYTE, readBuffer);
```

Obrázek 21. Zdrojový kód, který demonstruje čtení obrazových bodů z OpenGL ES povrchu.

5.4. Nahrávání videa v systému Android

Pro nahrávání videa v systému Android existuje několik možností. První je nahrávání videa z kamery zařízení pomocí systémové třídy *MediaRecorder* [31]. Další způsoby záznamu videa se dělí podle dvou kritérií. První je způsob získávání snímků z kamery (viz Kapitola 5.3), druhé kritérium definuje určuje použitou třídu/knihovnu pro enkódování videa. V práci jsou uvažovány dvě: třída *MediaCodec* systému Android a knihovna pro zpracování multimédií FFmpeg.

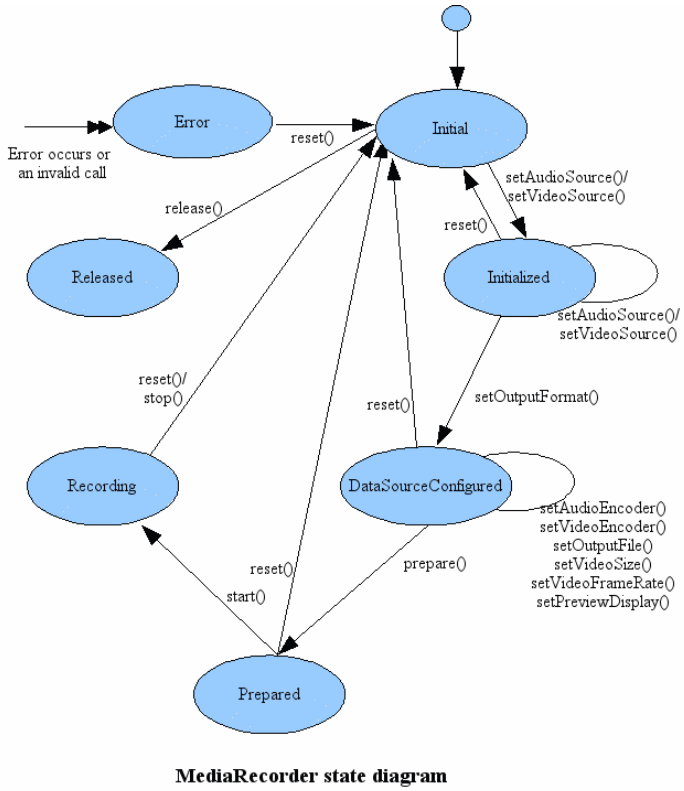
Všechny možnosti uvažují pouze nahrávání videa bez zvuku. Důvodem je implementace hlasového ovládání (viz Kapitola 6.4), které obsazuje kanál zvukového vstupu a bylo by neproveditelné nebo jen velmi obtížné zároveň nahrávat video se zvukem a ovládat zařízení hlasem. Zvukový doprovod videa není v tomto případě důležitější než možnost kombinace rozpoznání hlasu a záznamu videa.

5.4.1. Třídy/knihovny pro enkódování videa

5.4.1.1. MediaRecorder

Systémová třída MediaRecorder [31] umožňuje nahrávání videa, kde zdrojem je kamera zařízení. Spolu s videem může být nahrávaný také zvuk z mikrofonu zařízení.

Chování a způsob nastavení třídy je možné popsat stavovým automatem (viz Obrázek 22). Způsob inicializace a spuštění nahrávání se tak řídí přesnými pravidly a je nutné dodržovat pořadí volaných příkazů během konfigurace.



Obrázek 22. Stavový automat znázorňující ovládání třídy MediaRecorder. Reference: [31]

Konfigurace, spuštění nahrávání a jeho následné zastavení je popsáno úkázkou zdrojového kódu na Obrázku 23. Po vytvoření objektu *MediaRecorder* je nutné získat přístup

ke kameře zařízení metodou *Camera.unlock()*, to zajistí, že žádný jiný proces nemůže kameru využívat [22]. Následně je možné nastavit kameru metodou *setCamera()* a nastavit zdroj nahrávání metodou *setVideoSource()*. Následuje nastavení profilu nahrávání (viz Kapitola 5.4.1.1), určení výstupního souboru užitím *setOutputDisplay()* a nastavení, kam se má nahrávané video zobrazovat, pomocí metody *setPreviewDisplay()*.

Potvrzení konfigurace se provede metodou *prepare()* a spuštění samotného nahrávání videa pomocí *start()*. Od této chvíle je vstup z kamery ukládán do definovaného souboru dokud není zavolána metoda *MediaRecorder.stop()*. Dojde k zastavení nahrávání a ukončení výstupního souboru. Pro ukončení práce s objektem *MediaRecorder* je nutné zavolat metody *reset()* a *release()*, kdy dojde k uvolnění všech prostředků.

```
1   private SurfaceView surfaceView;
2   private Camera camera;
3   // ...
4      MediaRecorder mediaRecorder = new MediaRecorder();
5      // Vyber zdroje pro nahravani
6      this.camera.unlock();
7      this.mediaRecorder.setCamera(this.camera);
8      this.mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
9      // Nastaveni profilu nahravani
10     CamcorderProfile profile = CamcorderProfile.get(
11                                        CamcorderProfile.QUALITY_1080P);
12     // Nedokumentovana funkce umoznujici nahravat bez zvuku
13     profile.quality = CamcorderProfile.QUALITY_TIME_LAPSE_1080P;
14     this.mediaRecorder.setProfile(profile);
15     // Nastaveni vystupu nahravani
16     this.mediaRecorder.setOutputFile(this.getNewFilename());
17     this.mediaRecorder.setPreviewDisplay(surfaceView.getHolder().getSurface());
18     // Priprava MediaRecorder ka spusteni nahravani
19     try { this.mediaRecorder.prepare();
20     } catch (Exception e) { this.camera.lock(); return; }
21     // Spusteni nahravani
22     this.mediaRecorder.start();
23     // ...
24     // Ukonceni nahravani
25     try { mediaRecorder.stop();
26     } catch (IllegalStateException ise) { /* ... */ }
27     mediaRecorder.reset();
28     mediaRecorder.release();
29  // ...
```

Obrázek 23. Zdrojový kód, který popisuje způsob nahrávání videa z kamery pomocí třídy MediaRecorder.

Záznam je prováděn dle nastaveného profilu, který určuje rozlišení, datový tok a snímkovou frekvenci. Pro uložení dat je použit multimediální kontejner MP4 (viz Kapitola 5.2.9.2), pro enkódování videa se používá kodek H.264/MPEG-4 AVC (viz Kapitola 5.2.8.1).

Nahrávání bez zvuku Třída *MediaRecorder* standardně nepředpokládá nahrávání videa, které neobsahuje zvuk. Samotné vynechání volání funkce *setAudioSource()* nebylo spolehlivé a mohlo vést k chybě[1]. Proto byl zvolen způsob, kdy byl vygenerován profil nahrávání metodou *CamcorderProfile.get()*[32] a následně byla proměnná *quality* objektu *Profile* nastavena na jeden z typů *time lapse* videa. Jedná se o typ časosběrného videa (velmi nízká snímková frekvence), které neobsahuje zvuk, a je ho

[1]Tato metoda byla testována na zařízeních Samsung Galaxy Nexus (Android verze 4.2.1) a HTC Sensation (Android verze 4.0.2). Zařízení Samsung Galaxy Nexus takovýmto způsobem nahrávalo video bez zvuku, u HTC Sensation došlo k chybě a nebylo nahráno nic.

možné zvolit hodnotou *QUALITY_TIME_LAPSE_1080P* nebo jakoukoli jinou *QUALITY_TIME_LAPSE_**.

Prozkoumáním zdrojového kódu třídy *MediaRecorder* bylo zjištěno, že nastavení proměnné *quality* až po vygenerování požadovaného objektu *Profile* má vliv právě jen na přidání/nepřidání zvukové stopy do výsledného souboru, ale video je nahráváno podle původně zvoleného profilu a ne dle *time lapse* profilu. Předpokladem je, že vygenerovaný video profil bude pro standardní typ videa (např. *QUALITY_1080P* nebo jiný *QUALITY_**).

5.4.1.2. FFmpeg

FFmpeg[33] je knihovna pro zpracování multimediálních dat dostupná pod licencí GNU Lesser General Public License (LGPL) version 2.1.[34]. Knihovna poskytuje řadu dílčích vývojářských knihoven, které umožňují zpracování multimédií na úrovni dekódování a enkódování videa a audia, zpracovávání multimediálních kontejnerů, renderování a ukládání multimediálního obsahu, filtrování, a změnu parametrů. Knihovny jsou následující: libavutil, libavcodec, libavformat, libavdevice, libavfilter, libswscale, libswresample. Z pohledu použití v této práci je nejdůležitější součástí FFmpeg knihovna *libavcodec*, která umožňuje enkódování videa, knihovna *libavformat* pro ukládání multimediálních kontejnerů a knihovna *libswscale* pro změnu velikosti obrazu a barevného prostoru.

Následující ukázky zdrojového kódu byly testovány s FFmpeg verze 1.0, zkompilované pro systém Android a dostupné na [35]. Použitý kód je napsán v jazyce C++ a je inspirován implementacemi z [36], [37] a [38]. Tato část práce týkající se knihovny FFmpeg vznikla již na začátku roku 2013, avšak jako ukázka možného způsobu nahrávání a demonstrace jeho hlavních nevýhod (viz 5.4.2.2 a 5.4.2.3) je použitelná.

Incializace nahrávání Zdrojový kód popisující inicializaci nahrávání videa pomocí knihovny FFmpeg při komprimaci kodekem MPEG-4 do multimediálního kontejneru MP4 se nachází na Obrázku 24. Inicializace struktur, které jsou potřebné pro vytvoření videa se provede pomocí *av_register_all()*, následuje inicializace a nastavení struktury *AVOutputFormat* funkcí *av_guess_format()*, který zajišťuje definici výstupního formátu multimediálního souboru. Pro výstupní operace je použita struktura *AVFormatContext*, která je inicializována funkcí *avformat_alloc_context()*. Struktura obalující použitý kodek *AVCodec* se vytvoří při úspěšném nalezení kodeku v systému funkcí *avcodec_find_encoder()*, následně je vytvořena struktura multimediální stopy *AVStream* funkcí *avformat_new_stream()*. Pro definici parametrů kodeku, a tedy i samotného procesu enkódování, je použita struktura *AVCodecContext* inicializována při vytvoření multimediální stopy. Voláním funkce *avcodec_open2()* za použití parametrů *AVCodecContext* a *AVCodec* dojde k inicializaci samotného kodeku s definovaným nastavením.

Předpokládá se, že vstupní snímek určený pro enkódování videa nebude ve správném barevném formátu a rozlišení, které je vyžadováno. Z tohoto důvodu jsou inicializovány dvě struktury *AVFrame* pro provedení uvedeného převodu pomocí instance struktury *SwsContext*. Poslední operací inicializace je otevření výstupního souboru pro zápis funkcí *avio_open()* a samotný zápis hlavičky souboru funkcí *avformat_write_header()*.

Zpracování snímku Samotné zpracování snímku je demonstrováno ukázkou zdrojového kódu na Obrázku 25. Předpokládá se, že nejprve byla provedena inicializace sou-

```
1  // Jmeno vystupniho souboru
2  char *filename;
3  // Rozliseni vstupnich a vystupnich snimku
4  int srcW, srcH, dstW, dstH;
5  // Datova tok vysledneho videa
6  int videoBitrate = 1500000;
7  // Snimkovaci frekvence vystupniho videa
8  int fps = 25;
9  // ...
10 // Inicializace knihovny libavformat
11 av_register_all();
12 // Struktura definujici vystupni format video souboru
13 AVOutputFormat *of;
14 of = av_guess_format("mp4", NULL, NULL);
15 of->video_codec = AV_CODEC_ID_MPEG4;
16 of->audio_codec = AV_CODEC_ID_NONE;
17 // Struktura obsluhujici vstupne/vystupni operace
18 AVFormatContext *oc;
19 oc = avformat_alloc_context();
20 oc->oformat = of;
21 // Struktura obalijici audio/video kodek
22 AVCodec *codec;
23 codec = avcodec_find_encoder(of->video_codec);
24 // Struktura multimedialni stopy
25 AVStream *videoStream;
26 videoStream = avformat_new_stream(oc, codec);
27 // Struktura parametru kodeku
28 AVCodecContext *c;
29 c = videoStream->codec;
30 c->codec_id = of->video_codec;
31 c->codec_type = AVMEDIA_TYPE_VIDEO;
32 c->bit_rate = videoBitrate;
33 c->width = dstW;
34 c->height = dstH;
35 c->time_base = (AVRational){1, fps};
36 c->pix_fmt = outputPixelFormat;
37 // Inicializace kodeku codec pro dany kontext c
38 avcodec_open2(c, codec, NULL);
39 // Struktura pro snimek prevedeny do pozadovaneho rozliseni a barevneho
       prosturu
40 AVFrame *frame = avcodec_alloc_frame();
41 frame->pts = 0;
42 // Alokace pameti pro snimek prevedeny do pozadovaneho rozliseni a
       barevneho prosturu
43 uint8_t *frameBuf = (uint8_t *)av_malloc(avpicture_get_size(c->pix_fmt,
       c->width, c->height));
44 // Struktura pro vstupni snimek
45 AVFrame *frameTemp = avcodec_alloc_frame();
46 // Struktura pro konverzi barevnych formatu a obrazovou interpolaci
47 SwsContext *swsContext = sws_getCachedContext(swsContext, srcW, srcH,
       inputPixelFormat, c->width, c->height, c->pix_fmt, SWS_BILINEAR, NULL,
       NULL, NULL);
48 // Vytvoreni vystupniho souboru a zapis multimedialni hlavicky
49 strcpy(oc->filename, filename);
50 av_dump_format(oc, 0, filename, 1);
51 avio_open(&oc->pb, filename, AVIO_FLAG_WRITE);
52 avformat_write_header(oc, NULL);
```

Obrázek 24. Inicializace nahrávání videa pomocí knihovny FFmpeg. Vychází z [36]

částí knihovny FFmpeg (viz Obrázek 24). Vstupní obrazová data jsou v poli jbyte[2] *data*, vstupní a výstupní barevné prostory snímků jsou specifikovány ve struktuře *PixelFormat*.

Zpracování snímku začíná voláním funkce *avpicture_fill()*, které nejprve vloží do struktury *AVFrame* vstupní obrazová data, při druhém volání přiřadí další struktuře

[2]typ Java byte pole používaný v JNI - viz 4.1.2.1

```
1  // Surova data vstupni ho snimku
2  jbyte *data;
3  // Struktura barevneho prostoru vstupniho a vystupniho snimku
4  PixelFormat inPixFmt, outPixFmt;
5  // Velikost zapisovaciho bufferu
6  int outBufferSize = 256*1024;
7  // Alokace pameti pro vystupni operace
8  uint8_t *outbuf = (uint8_t *)av_malloc(outBufferSize);
9  // Nastaveni datovych poli pro vstupni snimek a snimek prevedeny do
       pozadovaneho rozliseni a barevneho prosturu (funkce sws_scale()).
10 avpicture_fill((AVPicture*)frameTemp, (uint8_t*)data, inPixFmt, srcW,
       srcH);
11 avpicture_fill((AVPicture*)frame, frameBuf, outPixFmt, c->width,
       c->height);
12 sws_scale(swsContext, frameTemp->data, frameTemp->linesize, 0, srcH,
       frame->data, frame->linesize);
13 // Struktura pro vystupni video paket
14 AVPacket videoPacket;
15 av_init_packet(&videoPacket);
16 videoPacket.data = outbuf;
17 videoPacket.size = outBufferSize;
18 // Provedeni enkodovani videa
19 int *gotPacket = new int[1];
20 int err = avcodec_encode_video2(c, &videoPacket, frame, gotPacket);
21 // Inkrementace casove znacky vstupniho snimku
22 frame->pts = frame->pts + 1;
23 // Pripadna oprava casove znacky a ulozeni video paketu do souboru
24 if (gotPacket[0] != 0) {
25    if (videoPacket.pts != AV_NOPTS_VALUE) {
26       videoPacket.pts = av_rescale_q(videoPacket.pts, c->time_base,
             videoStream->time_base);
27    }
28    if (videoPacket.dts != AV_NOPTS_VALUE) {
29       videoPacket.dts = av_rescale_q(videoPacket.dts, c->time_base,
             videoStream->time_base);
30    }
31    videoPacket.stream_index = videoStream->index;
32    int err = av_write_frame(oc, &videoPacket);
33 }
```

Obrázek 25. Zpracování video snímku pomocí knihovny FFmpeg. Vychází z [36]

AVFrame datové pole pro výstup obrazových dat po převodu barevného prostoru a změně rozlišení snímku, které je provedeno voláním funkce *sws_scale()* na obou zmíněných strukturách. Struktura *frame* poté obsahuje převedená data, která budou komprimována kodekem. Výstupem z kodeku je video paket obsažený ve struktuře *AVPacket*, ta je inicializována voláním *av_init_packet()*. Samotné enkódování je provedeno funkcí *av_encode_video2()*, která naplní video paket komprimovanými obrazovými daty. Paket je následně zapsán do souboru funkcí *av_write_frame()*.

```
1  // Uzavreni multimedialniho souboru
2  av_write_trailer(oc);
3  avio_close(oc->pb);
4  // Uvolneni vsech pouzitych struktur
5  delete[] gotPacket;
6  av_free(frameBuf);
7  av_free(frame);
8  av_free(frameTemp);
9  av_free(outbuf);
10 avcodec_close(c);
11 av_free(c);
12 av_free(oc);
```

Obrázek 26. Ukončení nahrávání videa pomocí knihovny FFmpeg. Vychází z [36]

Ukončení Na Obrázku 26 je uveden zdrojový kód, který provádí ukončení enkódování videa. Po zapsání všech snímků je nutné soubor ukončit a uzavřít, to se provede voláním funkcí *av_write_trailer()* a *avio_close()*. Následně je nutné uvolnit všechny prostředky, které byly během enkódování alokovány. To je provedeno voláním *delete[]*, *av_free()*, případně *avcodec_close()* při uvolňování prostředků kodeku. Uložený soubor je následně možné přehrát.

5.4.1.3. MediaCodec

Systémová třída *MediaCodec*[39], přidaná v API verze 16 (Android 4.1), umožňuje přístup k systémovým kodekům, které operují na nižší systémové hladině a existuje tak předpoklad, že díky tomu bude jejich výkon vyšší než v případě kodeků, které operují na uživatelské úrovni vzhledem k architektuře systému Android (viz Kapitola 5.4.1.2).

```
1   String MIME_TYPE = "video/avc";
2   MediaFormat format;
3
4   MediaCodec mediaCodec = MediaCodec.createEncoderByType(MIME_TYPE);
5   mediaCodec.configure(format, null, null, MediaCodec.CONFIGURE_FLAG_ENCODE);
6   mediaCodec.start();
7
8   ByteBuffer[] inputBuffers = mediaCodec.getInputBuffers();
9   ByteBuffer[] outputBuffers = mediaCodec.getOutputBuffers();
10
11  while (true) {
12     // ... Predani snimku kodeku ke zpracovani
13     // ... Odebrani a ulozeni zpracovanych snimku z kodeku
14  }
15
16  mediaCodec.stop();
17  mediaCodec.release();
18  mediaCodec = null;
```

Obrázek 27. Zdrojový kód zobrazuje základní použití MediaCodec. Vychází z [39]

Základní použití je zobrazeno na Obrázku 27. Kodek pro enkódování videa se vytvoří metodou *MediaCodec.createEncoderByType()*. Metodou *configure()* se provede konfigurace kodeku na základě zvoleného formátu (viz 5.4.1.3) a spustí se pomocí *start()*. Následně je možné získat přístup k vstupním a výstupním bufferům pomocí metody *getInputBuffers()* resp. *getOutputBuffers()*. Díky těmto bufferům je možné předávat kodeku snímky ke zpracování (viz 5.4.1.3) a následně také zpracované snímky z kodeku získat (viz 5.4.1.3). Po skončení práce s kodekem se zavolají metody *stop()* a *release()* pro uvolnění obsazených prostředků.

Vložení snímku ke zpracování kodekem se provádí po naplnění příslušného bufferu daty. Přístup k takovému bufferu získáme zavoláním metody *dequeueInputBuffer()*, která vrátí jeho identifikátor (index v poli *ByteBuffer* získaném voláním *getInputBuffers()*). Při volání metody se nastavuje čas, po který je možné čekat na buffer. Jedná se o ochranu proti dlouhému čekání v případě, že žádný ze vstupních bufferů není dostupný. Ve chvíli, kdy je buffer naplněn je možné ho předat kodeku ke zpracování. To se provádí metodou *queueInputBuffer()*, následně je buffer zařazen do fronty vstupních bufferů kodeku.

Předpokládá se, že vstupní snímek je ve správném barevném prostoru. Barevný prostor definujeme při vytváření formátu (viz Obrázek 30). Podpora barevného prostoru je závislá na konkrétním zařízení, avšak v případě použití kodeku H.264/MPEG-4 AVC

se předpokládá jeden z barevných prostorů YCbCR (viz 5.1.6.4) s použitím jednoho ze vzorkování (viz 5.1.6.4) a s jedním ze způsobů uložení (viz planární a semi-planární v 5.1.6.4).

```
long DEQUEUE_INPUT_BUFFER_TIMEOUT  = 10000; // 10 ms

// Predani snimku kodeku ke zpracovani
int inputBufferIndex =
    codec.dequeueInputBuffer(DEQUEUE_INPUT_BUFFER_TIMEOUT);
if (inputBufferIndex >= 0) {
   // ... Naplneni inputBuffers[inputBufferIndex] ...
   codec.queueInputBuffer(inputBufferIndex, ...);
}
```

Obrázek 28. Zdrojový kód zobrazuje předání snímku do MediaCodec. Vychází z [39]

Surface Druhou možností vložení snímku do enkodéru je použití systémové třídy *Surface* [40]. Ta funguje jako obálka pro obrazový buffer, kterou můžeme použít jako cíl vykreslování snímků z kamery pomocí OpenGL ES. Výhodou je, že celý proces vykreslování je řízen OpenGL ES, a tak je možné jednoduše aplikovat obrazové filtry a geometrické operace.

Vykreslovací povrch Surface získáme přímo z kodeku jako návratovou hodnotu volání funkce *MediaCodec.createInputSurface()*. Při nastavování barevného formátu v *MediaFormat* (viz Obrázek 30) je nutno v tomto případě použít hodnotu *MediaCodecInfo.CodecCapabilities.COLOR_FormatSurface* [41], která přímo určuje, že jako zdroj snímků bude použit povrch Surface. Díky tomu není nutné provádět převody barevného prostoru. Použití této metody zcela nahrazuje ruční vkládání snímků (viz Obrázek 28). Přesné použití této metody je popsáno v Kapitole 5.4.2.5.

```
long DEQUEUE_OUTPUT_BUFFER_TIMEOUT = 10000; // 10 ms

// Odebrani a ulozeni zpracovanych snimku z kodeku
ByteBuffer encodedData = null;
int outputBufferIndex = mediaCodec.dequeueOutputBuffer(bufferInfo,
    DEQUEUE_OUTPUT_BUFFER_TIMEOUT);
if (outputBufferIndex == MediaCodec.INFO_TRY_AGAIN_LATER) {
   // Nejsou k dispozici zadne zpracovane snimky
   break;
} else if (outputBufferIndex == MediaCodec.INFO_OUTPUT_BUFFERS_CHANGED) {
   // Doslo ke zmene vystupnich bufferu
   outputBuffers = mediaCodec.getOutputBuffers();
} else if (outputBufferIndex == MediaCodec.INFO_OUTPUT_FORMAT_CHANGED) {
   // Doslo ke zmene vystupnioho formatu ...
} else if (outputBufferIndex >= 0) {
   encodedData = outputBuffers[outputBufferIndex];
}

if ((bufferInfo.flags & MediaCodec.BUFFER_FLAG_CODEC_CONFIG) != 0) {
   // Buffer obsahuje inicializacni data
}

if ((bufferInfo.flags & MediaCodec.BUFFER_FLAG_SYNC_FRAME) != 0) {
   // Buffer obsahuje synchronizacni snimek
}

// ... ulozeni snimku z endodedData ...
mediaCodec.releaseOutputBuffer(outputBufferIndex, false);
```

Obrázek 29. Zdrojový kód zobrazuje získání snímku z MediaCodec. Vychází z [39]

Získání zpracovaného snímku z kodeku je nejčastěji následováno jeho uložením do některého z multimediálních kontejnerů. Snímek je uložený v bufferu, ke kterému získáme přístup zavoláním metody *dequeueOutputBuffer()*, která vrátí jeho identifikátor (index v poli *ByteBuffer* získaném voláním *getOutputBuffers()*). Při volání metody se nastavuje čas, po který je možné čekat na buffer, stejně jako v případě vkládání snímku ke zpracování. Následně se provede zpracování bufferu, například uložením do multimediálního kontejneru (viz 5.4.1.3). Pro zachování kontinuity zpracovávání kodekem je nutné vrátit buffer zpět kodeku metodou *queueOutputBuffer()* stejně jako v případě vstupních bufferů.

Ukončovací snímek je potřeba vytvořit v závěru enkódování. Voláním metody *MediaCodec.signalEndOfInputStream()* vyšleme kodeku signál k vytvoření ukončovacího snímku a ve chvíli, kdy kodek zpracuje všechny vložené snímky, vrátí také snímek ukončovací. Po volání této metody již není možné vkládat další snímky ke zpracování [39].

MediaFormat je třída systému Android, která umožňuje definici multimediálního formátu (viz Obrázek 30). Je používána při konfiguraci *MediaCodec* (viz Obrázek 27). Třída umožňuje definovat typ multimediálních dat (v tomto případě se jedná o MPEG-4 AVC - viz Kapitola 5.2.8.1), rozlišení (viz 5.1.2), barevný prostor (viz 5.1.6), datový tok (viz 5.1.5) snímkovou frekvenci (viz 5.1.1) a interval synchronizačního I-snímku (viz 5.2.7).

```
1  String MIME_TYPE = "video/avc";
2  MediaFormat format = MediaFormat.createVideoFormat(
3                       MIME_TYPE, videoWidth, videoHeight);
4  // Pri nahravani z EGL Surface: videoColorFormat =
5  // MediaCodecInfo.CodecCapabilities.COLOR_FormatSurface
6  format.setInteger(MediaFormat.KEY_COLOR_FORMAT, videoColorFormat);
7  format.setInteger(MediaFormat.KEY_BIT_RATE, videoBitRate);
8  format.setInteger(MediaFormat.KEY_FRAME_RATE, videoFrameRate);
9  format.setInteger(MediaFormat.KEY_I_FRAME_INTERVAL, videoIFrameInterval);
```

Obrázek 30. Zdrojový kód, který inicializuje MediaFormat.

```
1  MediaMuxer mediaMuxer;
2  BufferInfo bufferInfo = new MediaCodec.BufferInfo();
3
4  try { mediaMuxer = new MediaMuxer(filename,
5            MediaMuxer.OutputFormat.MUXER_OUTPUT_MPEG_4);
6  } catch (IOException e) { /* ... */ }
7
8  int trackIndex = mediaMuxer.addTrack(mediaCodec.getOutputFormat());
9  mediaMuxer.start();
10
11 // ...
12 // Zapis snimku do souboru
13 mediaMuxer.writeSampleData(trackIndex, encodedData, bufferInfo);
14 // ...
15
16 mediaMuxer.stop();
17 mediaMuxer.release();
```

Obrázek 31. Zdrojový kód, který zobrazuje základní použití MediaMuxer.

MediaMuxer je třída systému Android, která slouží k ukládání multimediálních dat do multimediálních kontejnerů (viz Kapitola 5.2.9.1). Je používána pro uložení enkódovaných video snímků do souboru (v tomto případě do kontejneru MP4 - viz Kapitola 5.2.9.2). Pro přidáni multimediální stopy se používá metoda *addTrack()*, vracející její identifikátor (viz Obrázek 31). V použitém případě je pro vytvoření potřeba znát formát multimediální stopy, který se získá metodou *MediaCodec.getOutputFormat()*. Vlastní zápis dat do souboru se provádí pomocí metody *writeSampleData()*, která je volána s argumentem identifikátoru stopy, samotnými daty a objektem *BufferInfo* [42], který nese informace o aktuálním snímku a je přepisován při získávání snímku z kodeku (viz Obrázek 29).

5.4.2. Metody nahrávání videa z kamery zařízení

Níže uvedený text popisuje uvažované metody pro nahrávání videa z kamery v systému Android při použití systémových funkcí nebo knihoven třetích stran. Kombinují se metody získávání snímků z 5.3 a metody pro kompresi videa z 5.4.1.

5.4.2.1. Systémové nahrávání

Pro použití systémového nahrávání je vyžadována pouze třída *MediaRecorder*, její použití je popsáno v Kapitole 5.4.1.1. Pomocí této třídy se ovládá celý proces nahrávání. Před jeho začátkem je nutné získat kontrolu nad kamerou zařízení. Poté se provede konfigurace nahrávání a spustí se samostatný záznam. Protože systém sám spouští běh třídy *MediaRecorder* a enkódování v samostatném vlákně, je její použití jednoduché a systémem je zajištěno, že proces bude mít dostatek prostředků.

Výhody

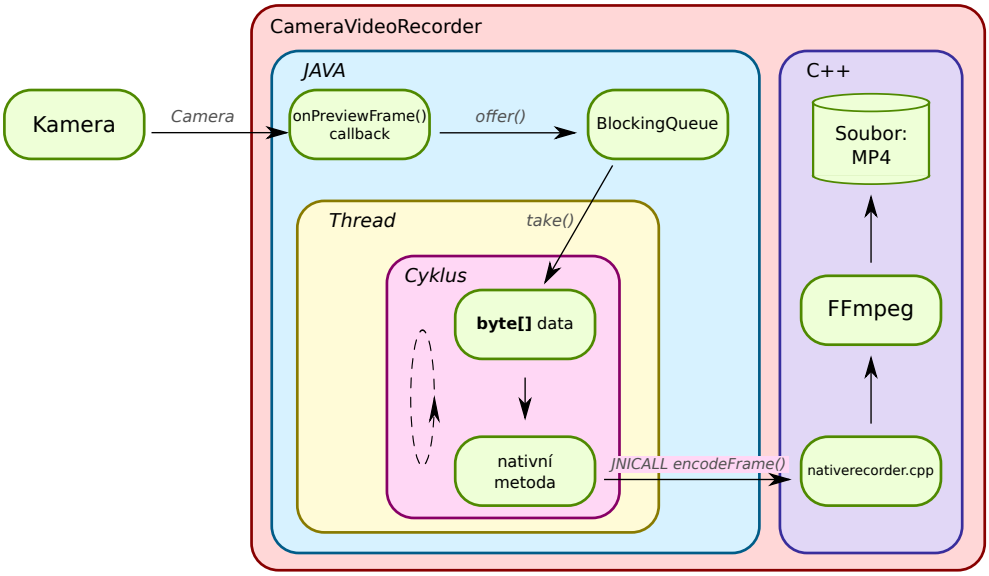
- Jedná se o systémovou funkci záznamu videa, dá se předpokládat, že bude fungovat správně na všech zařízeních s vestavěnou kamerou (viz 8.1.1).
- Využívá systémových kodeků pro kompresi videa, to zajišťuje optimální využití prostředků při kompresi.

Nevýhody

- Nemožnost jednotlivé snímky ani celé video jakkoli zpracovávat během nahrávání.
- Nahrávání je vázané na jeden výstupní soubor. Zastavení a opětovné spuštění nahrávání do nového souboru je provázeno několika vteřinovým výpadkem nahrávky během znovu nastavování nahrávání. Metoda tak není vhodná pro nahrávání v nekonečné smyčce (viz 5.5).
- Problémové nahrávání videa bez zvuku (viz 5.4.1.1).

5.4.2.2. FFmpeg a kopie snímků

Princip nahrávání videa pomocí knihovny FFmpeg je popsáno v kapitole 5.4.1.2 a získání kopie snímků je popsáno v kapitole 5.3.1. Výchozí barevný prostor kopie snímku je NV21 [23], video snímků YCbCr 4:2:0 (viz 5.2.8.1). Pro správnou funkčnost je nutné nastavit hodnotu barevného prostoru *PixelFormat inPixFmt* vstupních snímků na *PIX_FMT_NV21* definovanou v knihovně FFmpeg. Zároveň musí být adekvátně nastavena hodnota barevného prostoru *PixelFormat outPixFmt* výstupních video snímků H.264/MPEG-4 AVC videa na *PIX_FMT_YUV420P*. Obě tyto hodnoty jsou použity při převodu barevného prostoru a velikosti snímků (viz Obrázek 25).



Obrázek 32. Diagram znázorňující princip nahrávání videa z kamery pomocí FFmpeg a kopie snímků.

Princip je znázorněn na Obrázku 32. Kopie snímků, které jsou dodány callbackem od systému jsou postupně vkládány do *LinkedBlockingDeque*[43]. Jedná se o typ datové struktury blocking fronta, která umožňuje vícevláknový přístup pomocí zajištění čekání konzumujícího vlákna na naplnění prázdné fronty. Enkódování videa je prováděno v samostatném vlákně, které je spuštěno na úrovni Java Virtual Machine. Toto vlákno odebírá snímky z výše zmíněné *LinkedBlockingDeque* a zpracovává je dále pomocí volání nativních metod, které obsluhují knihovnu FFmpeg v jazyce C++.

Tato metoda **není implementována v aplikaci** Zoomera z důvodu nízkého přínosu a nekvalitního výstupu. V rámci práce však byla vytvořena jednoduchá aplikace FFmpegCamera (viz Přiložené CD - Příloha A.3), která funkci demonstruje. Výsledky testování této metody jsou v kapitole 8.1.3.

Výhody

- Kontrola nad procesem enkódování videa díky použití přímého přístupu ke kodeku v knihovně FFmpeg (viz Obrázek 25).
- Přístup k datům reprezentující jednotlivé snímky před i po enkódování.

Nevýhody

- Jakákoli úprava obrazu je implementačně náročná, protože se jednotlivé snímky nenacházejí v OpenGL ES, kde by bylo možné snadno aplikovat obrazové filtry a geometrické úpravy.
- Enkódování videa knihovnou FFmpeg je pomalé a proto přínosné jen pro nízká rozlišení výstupního videa (viz 8.1.3).
- Složitá implementace - nutnost provést všechna volání při inicializaci a enkódování pomocí FFmpeg (viz Obrázek 24, 25 a 26).

5.4.2.3. FFmpeg a glReadPixels

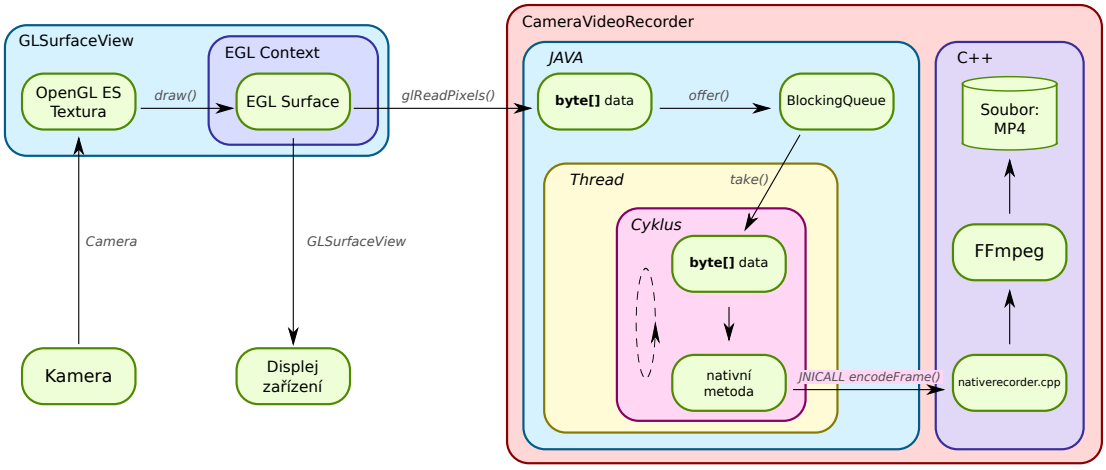
Princip nahrávání videa pomocí knihovny FFmpeg je popsáno v kapitole 5.4.1.2 a používání funkce *glReadPixels()* je popsáno v kapitole 5.3.3. Tato metoda je velmi po-

dobná metodě s použitím knihovny FFmpeg a kopie snímků (viz 5.4.2.2). Vstupem enkodéru však nejsou kopie snímků ze systémového callbacku, ale obrazová data získaná funkcí *glreadPixels()*, která čte obsah příslušného OpenGL ES povrchu. Metoda tak předpokládá, že snímky z kamery budou vykreslovány pomocí OpenGL ES. Způsob provedení tohoto vykreslování je popsán v [9]. Pro správnou funkčnost této metody je nutné, aby volání funkce *glReadPixels()* bylo prováděno ze stejného vlákna, ze kterého je vykreslován obraz pomocí OpenGL ES. Existence tohoto omezení je dána faktem, že OpenGL ES kontext, který obsahuje všechna obrazová data, je přístupný pouze z vlákna, ze kterého je vykreslován.

Na rozdíl od metody v 5.4.2.2 je barevný prostor vstupního snímku RGB, který je dán argumentem funkce *glReadPixels*. Proto je důležité nastavit hodnotu barevného prostoru *PixelFormat inPixFmt* vstupních snímků na *PIX_FMT_RGB*. Hodnota výstupního barevného prostoru *PixelFormat outPixFmt* zůstává stejná jako v 5.4.2.2.

Princip je znázorněn na Obrázku 33. Obrazová data jsou průběžně načítána funkcí *glReadPixels()* během vykreslování OpenGL ES povrchu a následně vkládána do *LinkedBlockingDeque*[43]. Enkódování videa je v samostatném vlákně, na úrovni Java Virtual Machine. Toto vlákno odebírá snímky z *LinkedBlockingDeque* a zpracovává je dále nativně v C++ pomocí knihovny FFmpeg.

Tato metoda **není implementována v aplikaci** Zoomera z důvodu nízkého přínosu, nekvalitního výstupu a pomalého zpracování. Byla testována pouze funkce *glReadPixels()*. Protože je přenos mezi grafickou pamětí a operační pamětí zařízení omezený z hlediska datové propustnosti, bylo vykonávání funkce *glReadPixels()* velmi pomalé. Z toho důvodu nebyla tato metoda nahrávání implementována vůbec, byl vytvořen pouze tento návrh. Výsledky testování této metody jsou v kapitole 8.1.2.



Obrázek 33. Diagram znázorňující princip nahrávání videa z kamery pomocí FFmpeg a glReadPixels().

Výhody

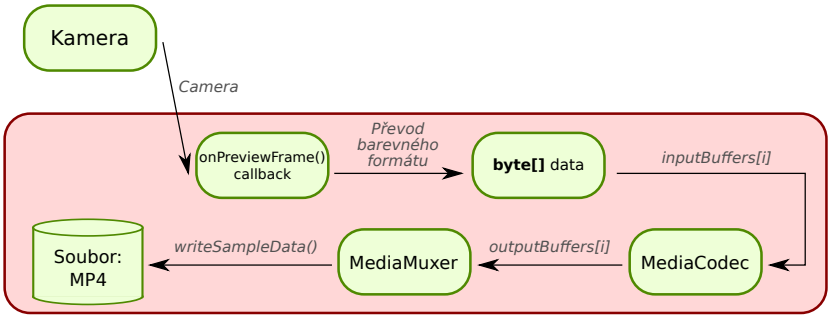
- Možnost efektivně upravovat snímky pomocí grafické knihovny OpenGL ES.
- Kontrola nad procesem enkódování videa díky použití přímého přístupu ke kodeku v knihovně FFmpeg (viz Obrázek 25).
- Přístup k datům reprezentující jednotlivé snímky před i po enkódování.

Nevýhody

- Velmi pomalé čtení obrazových dat funkcí *glReadPixels()* (viz 8.1.2).
- Enkódování videa knihovnou FFmpeg je pomalé a proto přínosné jen pro nízká rozlišení výstupního videa (viz 8.1.3).
- Složitá implementace - nutnost provést všechna volání při inicializaci a enkódování pomocí FFmpeg (viz 24, 25 a 26).

5.4.2.4. Kopie snímků a MediaCodec

Způsob nahrávání videa pomocí třídy MediaCodec je popsáno v kapitole 5.4.1.3, získávání kopií snímků je obsaženo v kapitole 5.3.1.



Obrázek 34. Diagram znázorňující princip nahrávání videa z kamery pomocí MediaCodec a kopie snímků.

Výchozí barevný prostor kopie snímku je NV21 [23], video snímky jsou v jednom z formátů YCbCr (viz Obrázek 28). Proto je nutné provést převod těchto formátů. Z tohoto důvodu nebyla tato metoda implementována a byla použita následující metoda.

Princip je znázorněn na Obrázku 34. Kopie snímků, které jsou dodány callbackem od systému jsou postupně převáděny do požadovaného barevného prostoru a následně vkládány do kodeku (viz Obrázek 28). Enkódování videa je prováděno instancí třídy *MediaCodec*, která využívá systémových nízkoúrovňových kodeků pro enkódování videa. Ve chvíli, kdy je snímek enkódovan, je systémem vrácen ve výstupních bufferech (viz Obrázek 29). Buffery jsou uloženy instancí *MediaMuxer* do souboru (viz Obrázek 31).

Tato metoda **není implementována v aplikaci** Zoomera, protože byla použita metoda OpenGL ES textura a *MediaCodec* (viz 5.4.2.5), která oproti této metodě nevyžaduje implementaci převodu barevného formátu a navíc umožňuje efektivně zpracovávat obraz přímo v OpenGL ES. Z tohoto důvodu je metoda obsažena pouze jako návrh.

Výhody

- Rychlé enkódování pomocí systémových kodeků (viz Kapitola 8.1.4).
- Díky rychlosti i přístupu k jednotlivým zpracovaným snímkům je tato metoda vhodná pro nahrávání v nekonečné smyčce.
- Přístup k datům reprezentující jednotlivé snímky před i po enkódování.

Nevýhody

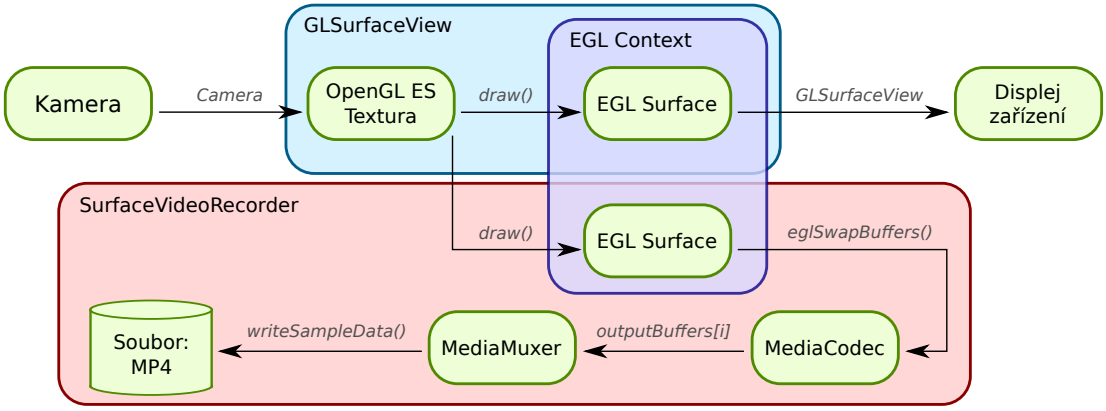
- Úprava obrazu je implementačně náročná, jednotlivé snímky se nenacházejí v OpenGL ES, které zjednodušuje použití obrazových filtrů a geometrických úprav.
- Nutnost provádět převod barevného prostoru z NV21 na podporovaný formát.
- Protože metoda využívá třídy *MediaCodec*, *MediaFormat* a *MediaMuxer*, je omezena na použití na zařízeních se systémem Android verze 4.3 a vyšší.

5.4.2.5. Textura v OpenGL ES a MediaCodec

Tato metoda nahrávání kombinuje použití vykreslování OpenGL ES textury s obrazem z kamery (viz 5.3.2) a systémovou třídu *MediaCodec* (viz 5.4.1.3). Princip metody je založen použití dvou vykreslovacích povrchů (EGL Surface), kde první bude přidělen k displeji zařízení tak, aby uživatel viděl obraz z kamery, a druhý bude přidělen kodeku jako zdroj obrazových dat pro vytvoření videozáznamu. Obrazová data jsou uložena v OpenGL ES textuře. Samotné renderování je prováděno dvakrát: poprvé na displej zařízení, podruhé mimo obrazovku na povrch kodeku.

Dokumentace systému Android nepopisuje způsob použití třídy *MediaCodec* pro enkódování videa z OpenGL ES, i když použité metody třídy jsou v dokumentaci zastoupeny. Pro vytvoření této metody nahrávání videa z kamery byly použity ukázky zdrojových kódů z [44] a [45], které slouží pro demonstraci funkčnosti třídy *MediaCodec* a popisují, jakým způsobem se se třídou pracuje. Autorem první ukázky je organizace The Android Open Source Project, druhé pak Google Inc..

Protože je generace a komprese snímků v tomto případě závislá na frekvenci vykreslování pomocí OpenGL ES, není možné zajistit přesnou snímkovou frekvenci. Z tohoto důvodu není ve spojitosti s touto metodou snímková frekvence uváděna. Protože ale jsou snímky videa ukládány s časovou značkou, tak i v případě, že dojde ke změně vykreslovací frekvence a tím ke ztrátě snímků, bude výsledné video přehratelné bez časového zkreslení.



Obrázek 35. Diagram znázorňující princip nahrávání videa z kamery pomocí OpenGL textury a třídy MediaCodec.

Princip je demonstrován na Obrázku 35. Implementace zahrnuje třídu *SurfaceVideoRecorder*, která nahrávání řídí. Ta je napojena na *GLSurfaceView*, které slouží k řízení vykreslování jak na obrazovku, tak na povrch kodeku. Obě instance spolu sdílí EGLContext, který uchovává stav OpenGL ES knihovny. Kodek zpracovává obrazová data získaná z EGLSurface a jeho výstupem jsou buffery obsahující enkódovaná data. Ta jsou zpracována instancí třídy MediaMuxer, která je uloží do MP4 kontejneru.

EGL - Native Platform Interface Rozhraní EGL slouží jako propojení mezi grafickou knihovnou OpenGL ES a nativním okenním systémem, který je vázaný na danou platformu. Rozhraní se stará o řízení přidělení paměťového prostoru během vykreslování, spravuje jednotlivé vykreslovací kontexty a řídí průběh a synchronizaci renderování.[46]

Třída *GLSurfaceView* [24], která je použita při zpracování a vykreslení obrazu z kamery (viz 5.3.2), řídí sama vytvoření EGL displeje, EGL kontextu a EGL povrchu, na

který se provádí samotné renderování. Pro nahrávání videa z kamery pomocí *MediaCodec* je však nutné vytvořit ze zmíněného EGL kontextu sdílený kontext a připojit k němu druhý EGL povrch, který bude sloužit jako zdroj obrazových snímků pro enkódování videa (viz Obrázek 35).

Inicializace sdíleného EGL kontextu Pro inicializaci a ovládání EGL byla vytvořena třída *EGLWrapper*, která obsahuje všechna volání spojená s EGL. Tato třída je součástí třídy *SurfaceVideoRecorder*, která slouží k nahrávání videa z OpenGL ES textury pomocí *MediaCodec*. Před začátkem nahrávání je nutné inicializovat sdílený EGL kontext (viz Obrázek 36), aby bylo možné vykreslovat snímky z kamery jak na obrazovku, tak na povrch kodeku.

```
1  // Typ povrchu urceny pro nahravani
2  int EGL_RECORDABLE_ANDROID = 0x3142;
3  // Struktury EGL urcene pro vykresleni na povrch kodeku
4  EGLDisplay mEGLDisplay = EGL14.EGL_NO_DISPLAY;
5  EGLContext mEGLContext = EGL14.EGL_NO_CONTEXT;
6  EGLSurface mEGLSurface = EGL14.EGL_NO_SURFACE;
7  // Struktury EGL urcene pro vykresleni na obrazovku
8  EGLDisplay mScreenEglDisplay = EGL14.EGL_NO_DISPLAY;
9  EGLContext mScreenEglContext = EGL14.EGL_NO_CONTEXT;
10 EGLSurface mScreenEglDrawSurface = EGL14.EGL_NO_SURFACE;
11 EGLSurface mScreenEglReadSurface = EGL14.EGL_NO_SURFACE;
12 // Vykreslovaci povrch kodeku
13 Surface encoderSurface = mediaCodec.createInputSurface();
14 // ...
15 // Nastaveni EGL struktur pro vykresleni na obrazovku
16 mScreenEglDisplay = EGL14.eglGetCurrentDisplay();
17 mScreenEglContext = EGL14.eglGetCurrentContext();
18 mScreenEglDrawSurface = EGL14.eglGetCurrentSurface(EGL14.EGL_DRAW);
19 mScreenEglReadSurface = EGL14.eglGetCurrentSurface(EGL14.EGL_READ);
20 // Nastaveni EGL displeje pro vykresleni na povrch kodeku
21 mEGLDisplay = EGL14.eglGetDisplay(EGL14.EGL_DEFAULT_DISPLAY);
22 // Inicializace EGL
23 int[] version = new int[2];
24 EGL14.eglInitialize(mEGLDisplay, version, 0, version, 1);
25 // Nastaveni konfigurace EGL
26 int[] attribList = {EGL14.EGL_RED_SIZE, 8,  EGL14.EGL_GREEN_SIZE, 8,
       EGL14.EGL_BLUE_SIZE, 8, EGL14.EGL_ALPHA_SIZE, 0, EGL14.EGL_DEPTH_SIZE,
       16, EGL14.EGL_SAMPLE_BUFFERS, 0, EGL14.EGL_SAMPLES, 0,
       EGL14.EGL_RENDERABLE_TYPE, EGL14.EGL_OPENGL_ES2_BIT,
       EGL_RECORDABLE_ANDROID, 1, EGL14.EGL_NONE};
27 EGLConfig[] configs = new EGLConfig[1];
28 int[] numConfigs = new int[1];
29 EGL14.eglChooseConfig(mEGLDisplay, attribList, 0, configs, 0,
       configs.length, numConfigs, 0);
30 // Inicializace sdileneho EGL kontextu
31 int[] attrib_list = {EGL14.EGL_CONTEXT_CLIENT_VERSION, 2, EGL14.EGL_NONE};
32 mEGLContext = EGL14.eglCreateContext(mEGLDisplay, configs[0],
       mScreenEglContext, attrib_list, 0);
33 // Vytvoreni EGL povrchu a jeho pripojeni k vykreslovacimu povrchu kodeku
34 int[] surfaceAttribs = {EGL14.EGL_NONE};
35 mEGLSurface = EGL14.eglCreateWindowSurface(mEGLDisplay, configs[0],
       encoderSurface, surfaceAttribs, 0);
```

Obrázek 36. Zdrojový kód zobrazuje inicializaci EGL. Vychází z [44]

Pro možnost přepínání mezi kontexty, je nutné uložit základní EGL struktury inicializované instancí třídy *GLSurfaceView* do proměnných tak, abychom k nim měli přístup, i když je kontext přepnutý. Jedná se o struktury *EGLDisplay*, *EGLContext*, *EGLSurface* pro čtení a zápis. Následně získáme výchozí *EGLDisplay* pro sdílený kontext funkcí *EGL14.eglGetDisplay()*, který následně inicializujeme funkcí *eglInitialize()*. Funkcí *egl-*

ChooseConfig() zvolíme konfiguraci EGL pro nahrávání z OpenGL ES 2.0 povrchu, který je specifikovaný barevným prostorem RGB888 (8 bitů na každý kanál). Následně vytvoříme funkcí *eglCreateContext()* sdílený EGL kontext s použitím vytvořeného EGL displeje, konfigurace a EGL kontextu určeného pro vykreslování na obrazovku, který chceme sdílet. Posledním voláním *elgCreateWindowsSurface()* vytvoříme EGL povrch, který bude propojen s povrchem kodeku. Na něj budou vykreslovány snímky určené k enkódování.

Nahrání snímku Předpokladem je, že je aktivní vykreslování v instanci třídy *GLSurfaceView*, následně je nutné aby byla provedena inicializace sdíleného EGL kontextu. Poté se v každém cyklu OpenGL ES vykreslování postupně zobrazuje a zpracovává nový snímek z kamery.

Proces je popsán na Obrázku 37. Veškeré vykreslování předchází aktualizace textury novým snímkem z kamery (viz Obrázek 20). Následuje vykreslení snímku z kamery pomocí OpenGL ES na obrazovku zařízení. Poté voláním funkce *eglMakeCurrent()* změníme parametry sdíleného EGL kontextu tak, aby všechna následující volání OpenGL ES byla směřována na povrch kodeku. Opakuje se vykreslení snímku z kamery pomocí OpenGL ES, obraz je však v tomto případě renderován na povrch kodeku. Funkcí *eglPresentationTimeANDROID()* se nastaví časová značka pro aktuální snímek a funkcí *eglSwapBuffers()* se vykreslený snímek poskytne kodeku ke zpracování. Voláním *eglMakeCurrent()* změníme parametry sdíleného EGL kontextu a všechna následující OpenGL ES volání budou směrována na obrazovku zařízení. Protože vykreslování na obrazovku i na povrch kodeku provádí pomocí standardních volání knihovny OpenGL ES, je možné použití libovolného shader programu a docílit tak možnosti libovolně obraz upravovat.

```
1  // ... Obnoveni snimku v OpenGL ES texture.
2  // draw() ... Vykresleni snimku na obrazovku
3  // Nastavime struktury EGL pro kodek jako aktivni
4  EGL14.eglMakeCurrent(mEGLDisplay, mEGLSurface, mEGLSurface, mEGLContext);
5  // draw() ... Vykresleni snimku na povrch kodeku
6  // Nastaveni casove znacky snimku
7  EGLExt.eglPresentationTimeANDROID(mEGLDisplay, mEGLSurface,
       cameraSurface.getTimestamp());
8  // Vymenou obrazovych bufferu dojde k zpracovani vykresleneho snimku
       kodekem
9  EGL14.eglSwapBuffers(mEGLDisplay, mEGLSurface);
10 // Nastavime struktury EGL pro obrazovku jako aktivni
11 EGL14.eglMakeCurrent(mScreenEglDisplay, mScreenEglDrawSurface,
       mScreenEglReadSurface, mScreenEglContext);
```

Obrázek 37. Zdrojový kód zobrazuje vykreslení a zpracování jednoho snímku. Vychází z [44]

```
1  // Nastavime struktury EGL pro obrazovku jako aktivni
2  EGL14.eglMakeCurrent(mScreenEglDisplay, mScreenEglDrawSurface,
       mScreenEglReadSurface, mScreenEglContext);
3  // Odstranime vsechny EGL struktury pro kodek
4  EGL14.eglDestroySurface(mEGLDisplay, mEGLSurface);
5  EGL14.eglDestroyContext(mEGLDisplay, mEGLContext);
6  EGL14.eglTerminate(mEGLDisplay);
7  // Uvolnime vykreslovaci povrch kodeku
8  encoderSurface.release();
```

Obrázek 38. Zdrojový kód zobrazuje uvolnění EGL. Vychází z [44]

Ukončení nahrávání Po zpracování všech požadovaných snímků se zastaví vykreslování dalších a provede se změna parametrů sdíleného EGL displeje funkcí *eglMakeCurrent()* tak, aby vykreslování bylo cíleno na obrazovku. Jako poslední akce po ukončení nahrávání je nutné uvolnit použité EGL struktury (viz Obrázek 38).

Alternativa zdvojeného vykreslování Při nahrávání videa výše popsanou metodou je nutné vykreslovat snímek dvakrát. Poprvé při zobrazení na obrazovku, podruhé při vykreslování na povrch kodeku. Tento způsob by bylo možné nahradit pouze jedním vykreslováním, ale z důvodu nutnosti přepracovat celý způsob vykreslování a zásadně tak zasáhnout do jádra aplikace Zoomera nebylo toto řešení implementováno.

```
1  // Vykreslovaci povrch obrazovky
2  Surface displaySurface;
3  // Vykreslovaci povrch kodeku
4  Surface encoderSurface;
5  // Struktury EGL urcene pro vykresleni na obrazovku
6  EGLDisplay mScreenEglDisplay;
7  EGLContext mScreenEglContext;
8  EGLSurface mScreenEglSurface;
9  // Struktury EGL urcene pro vykresleni na povrch kodeku
10 EGLDisplay mEGLDisplay;
11 EGLContext mEGLContext;
12 EGLSurface mEGLSurface;
13 // Rozliseni videa
14 int videoWidth, videoHeight;
15 // ...
16 // draw() ... Vykresleni snimku na displaySurface
17 // Nastavime struktury EGL pro kodek jako aktivni. Jako povrch pro cteni
       nastavime povrch obrazovky, kde je jiz vykreslen snimek.
18 EGL14.eglMakeCurrent(mEGLDisplay, mScreenEglSurface, mEGLSurface,
       mEGLContext);
19 // Zkopirujeme data z aktualiniho EGL povrchu pro cteni (povrch obrazovky)
       do aktualniho EGL povrchu pro zapis (povrch kodeku).
20 GLES30.glBlitFramebuffer(0, 0, displaySurface.getWidth(),
       displaySurface.getHeight(), 0, 0, videoWidth, videoHeight,
       GLES30.GL_COLOR_BUFFER_BIT, GLES30.GL_NEAREST);
21 // Nastaveni casove znacky snimku
22 EGLExt.eglPresentationTimeANDROID(mEGLDisplay, mEGLSurface,
       cameraSurface.getTimestamp());
23 // Vymenou obrazovych bufferu dojde k zpracovani vykresleneho snimku
       kodekem
24 EGL14.eglSwapBuffers(mEGLDisplay, mEGLSurface);
25 // Nastavime struktury EGL pro obrazovku jako aktivni
26 EGL14.eglMakeCurrent(mScreenEglDisplay, mScreenEglSurface,
       mScreenEglSurface, mScreenEglContext);
27 // Provede se vykresleni z EGL povrchu obrazovky na obrazovku zarizeni
28 EGL14.eglSwapBuffers(mScreenEglDisplay, mScreenEglSurface);
```

Obrázek 39. Použití funkce *GLES30.glBlitFramebuffer()* jako alternativa zdvojenému vykreslování. Vychází z [45]

Řešení uvažuje použití funkce *GLES30.glBlitFramebuffer()* (viz Obrázek 39), která umožňuje zkopírování obrazových dat z aktuálního EGLSurface určeného pro čtení do aktuálního EGLSurface určeného pro zápis. Toho je použito tak, že je nejdříve vykreslen obraz na EGL povrch obrazovky, poté je změněn EGL kontext na kontext kodeku a použitím zmíněné funkce dojde k zkopírování dat z EGL povrch obrazovky na EGL povrch kodeku. Odpadá tak zdvojené vykreslování.

Řešení vyžaduje řízení EGL struktur pro obrazovku i pro kodek. Tím však není možné použít řešení s *GLSurfaceView* a bylo by nutné navrhnout celý nový systém vykreslování v aplikaci. Protože aktuální řešení funguje správně, nebyl tento návrh implementován.

Výhody

- Možnost efektivně upravovat jednotlivé snímky pomocí OpenGL ES.
- Rychlé enkódování pomocí systémových kodeků (viz Kapitola 8.1.4).
- Díky rychlosti i přístupu k jednotlivým zpracovaným snímkům je tato metoda vhodná pro nahrávání v nekonečné smyčce.
- Přístup k datům reprezentující jednotlivé snímky po enkódování.

Nevýhody

- Vykreslování probíhá dvakrát. Poprvé na obrazovku, podruhé na EGL povrch kodeku.
- Snímky jsou posílány z kamery přímo do textury v OpenGL ES. Není tak možné touto metodou získat obrazová data před enkódováním.
- Pro funkčnost této metody je nutné provést inicializaci EGL ve sdíleném kontextu s GLSurfaceView.
- Protože metoda využívá třídy *MediaCodec*, *MediaFormat* a *MediaMuxer*, je omezena na použití na zařízeních se systémem Android verze 4.3 a vyšší.

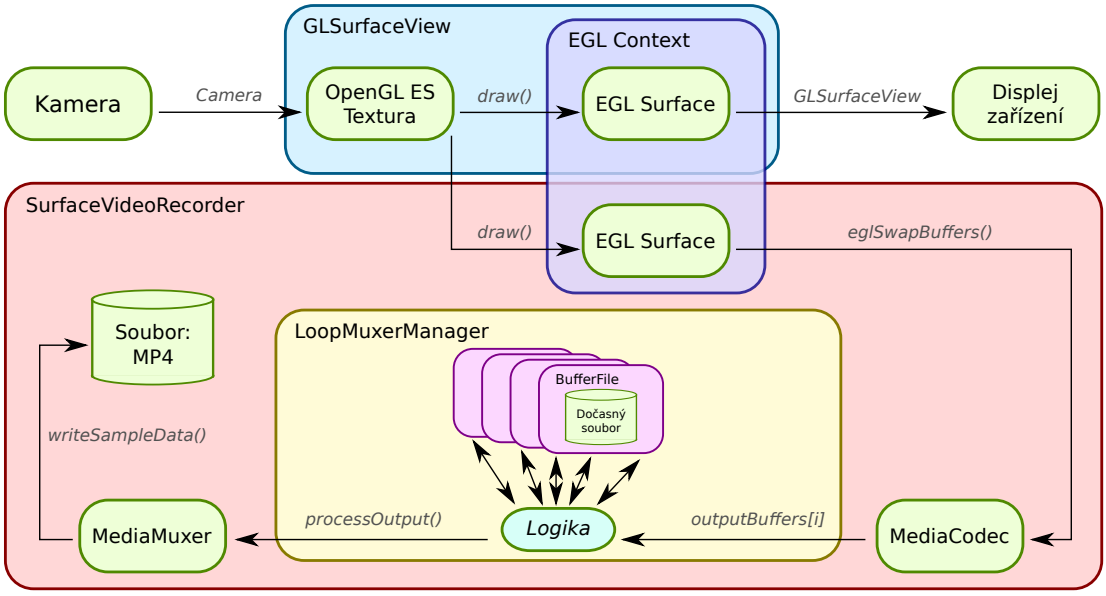
5.5. Nahrávání v nekonečné smyčce

Nahráváním videa v nekonečné smyčce je myšlen takový způsob záznamu, kdy je video zachytáváno nepřetržitě od začátku nahrávání. Kdykoli během záznamu je proces zastaven, je k dispozici interval posledních n časových jednotek záznamu. Zároveň jsou data spadající již mimo tento interval během celého nahrávání průběžně mazána. Prostor, který obsazuje nahrávané video, se tak zvětšuje pouze do určité meze, která je dána požadovanou délkou n dostupného intervalu záznamu a na niž nemá samotná doba nahrávání vliv.

5.5.1. Princip

Nahrávání v nekonečné smyčce je založeno na principu ukládání enkódovaných obrazových snímků do několika dočasných souborů, které mají pevnou velikost jsou postupně mazány od nejstaršího tak, aby bylo zajištěno, že se na disku/paměťové kartě zařízení nachází posledních m vteřin záznamu. Velikost m je po po celou dobu nahrávání větší nebo rovna požadované délce n video smyčky, výjimku tvoří jen prvních n vteřin záznamu, kdy ještě nebyl nahrán video úsek o délce požadované velikost smyčky. Velikost m je shora omezena průměrnou délkou záznamu, který se umístí do jednoho dočasného souboru násobený počtem dočasných souborů, které je potřeba pro uložení celé smyčky.

Pokud omezíme velikost jednoho dočasného souboru například na zapsání 750 snímků (při 25 fps - 30 s záznamu) a budeme požadovat 45 s smyčku, bude nejdelší uložený záznam obsahovat posledních 60 s a bude uložen nejvíce ve dvou souborech. Tento předpoklad platí v případě, že nedojde ke ztrátě nebo zpoždění nějakého snímku. Pokud k němu dojde, platí stále maximální datová velikost souboru (je omezena na počet snímků), ale v souborech bude uložený delší časový úsek vzhledem k časovým značkám snímků.



Obrázek 40. Diagram znázorňující princip nahrávání videa z kamery v nekonečné smyčce pomocí OpenGL textury a třídy MediaCodec.

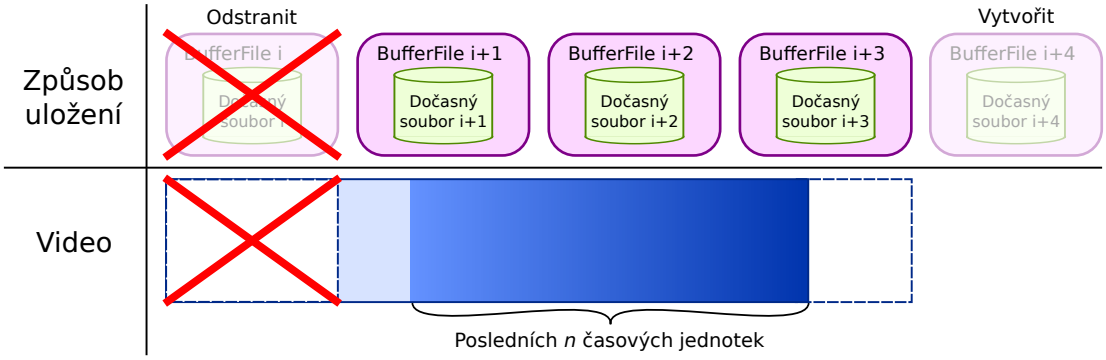
Implementace nahrávání videa v nekonečné smyčce rozšiřuje funkčnost třídy *SurfaceVideoRecorder*, která je určená pro nahrávání videa z kamery pomocí OpenGL ES textury a třídy *MediaCodec* (viz 5.4.2.5). Tato metoda nahrávání videa byla vybrána

pro implementaci nahrávání ve smyčce, protože jako jediná z uvažovaných metod (viz 5.4.2) umožňovala jak kontrolu nad každým snímkem, tak vykreslování pomocí OpenGL ES, které umožní úpravu videa v reálném čase během nahrávání.

Princip metody je znázorněn na Obrázku 40. Záznam, úprava a enkódování snímků probíhá zcela shodně jako standardní nahrávání touto metodou (viz 5.4.2.5). Rozdíl nastává až při zpracování výstupu z objektu *MediaCodec*, kde jednotlivé snímky nejsou ihned ukládány do MP4 kontejneru objektem *MediaMuxer*, ale jsou zpracovávány objektem implementované třídy *LoopMuxerManager*. Ten slouží jako správce nahraných snímků, ukládá je do dočasných souborů a v případě, že již nejsou zapotřebí (nacházejí se mimo smyčku), jsou smazány.

5.5.2. Dočasné ukládání

Pro dočasné ukládání byla vytvořena třída *BufferFile*, která slouží k ukládání a načítání video snímků do/ze souboru a je ovládána objektem *LoopMuxerManager*. S třídou je spojen dočasný soubor na disku/paměťové kartě zařízení, který slouží pro ukládání jednotlivých snímků. Třída *BufferFile* nese informace o každém obsaženém snímku. Mezi informace patří časová značka snímku, velikost bufferu snímku a typ snímku (viz 5.2.7). *LoopMuxerManager* zapisuje vkládané snímky do nejnovějšího dočasného souboru, dokud nedojde k jeho zaplnění. K zaplnění dojde při vložení 750 video snímků. Takový limit zaručuje, že v případě krátké smyčky se nebude vytvářet zbytečně velký dočasný soubor a naopak v případě dlouhé smyčky nebude dočasných souborů příliš velké množství.



Obrázek 41. Diagram znázorňující způsob uložení video dat v dočasných souborech *BufferFile* a jejich odpovídající úsek videozáznamu.

LoopMuxerManager drží seznam všech dočasných souborů, v případě, že se nejnovější dočasný soubor zaplní, je vytvořen nový dočasný soubor. Při vytváření nového dočasného souboru se kontroluje, zda seznam neobsahuje dočasný soubor, který nese pouze snímky, které jsou z hlediska video smyčky již nepotřebné. Pokud je takový soubor nalezen, je odebrán ze seznamu a následně smazán. Princip uložení videozáznamu do dočasných souborů je popsán na Obrázku 41.

5.5.3. Vytvoření video souboru

Jakmile je nahrávání ve smyčce zastaveno, spustí *LoopMuxerManager* zpracování dočasných souborů za účelem vytvoření výsledného video souboru, který obsahuje video úsek se záznamem posledních n časových jednotek. Vypočte se počáteční čas, ve kterém by měl záznam ve smyčce začínat (od času posledního nahraného snímku se odečte požadovaná délka smyčky).

V seznamu dočasných souboru *BufferFile* se vyhledá ten, který obsahuje nejstarší část záznamu. V tomto souboru se nalezne synchronizační snímek (viz I-frame 5.2.7), který má nejbližší nižší čas, než je vypočtený počáteční čas. Je nutné hledat pouze synchronizační snímky, protože takový snímek obsahuje všechna obrazová data (ne jen rozdílová data), a tak je možné, aby jím video začínalo a nebylo poškozené. Od tohoto místa se začnou číst data a ukládat do MP4 souboru pomocí třídy *MediaMuxer*.

Díky znalosti velikosti dat jednotlivých snímků je možné ve zpracovávaném dočasném souboru nalézt pozici, kde začíná požadovaný snímek a od tohoto místa začít číst data jednotlivých snímků. Data jsou uložena do byte pole a vložena do *ByteBuffer*[28] metodou *ByteBuffer.wrap()*. Dále jsou aktualizovány hodnoty v objektu *BufferInfo*[42] podle aktuálního snímku a spolu s daty snímku je předán objektu *MediaMuxer* metodou *writeSampleData()*, který je zapíše do výsledného MP4 souboru (viz Obrázek 31). Takto jsou zpracovány všechny snímky ve všech příslušných dočasných souborech.

5.6. Dodatky k implementaci v aplikaci Zoomera

Nehledě na množství prozkoumaných metod nahrávání, byly do aplikace Zoomera implementovány jen ty, které mají pro uživatele potenciální přínos ať z hlediska výkonu při zpracování, funkcí či možností úprav obrazu. Protože aplikace využívá třídy *MediaCodec*, *MediaFormat* a *MediaMuxer*, je omezena na použití na zařízeních se systémem Android verze 4.3 a vyšší.

5.6.1. Nahrávání videa

Ze všech uvedených metod byly vybrány dvě, které byly implementovány do aplikace Zoomera. První je metoda systémového nahrávání, druhá je metoda použití OpenGL ES textury a třídy *MediaCodec*. Nahrané video se uloží do složky */sdcard/Zoomera/* s názvem souboru video-*datum*-*čas*.mp4.

5.6.1.1. Systémové nahrávání

Použití systémového nahrávání je popsáno v 5.4.2.1. Možnosti nastavení nahrávání v grafickém uživatelském rozhraní jsou popsány v Kapitole 7.4.1 v závislosti na podporovaných video profilech (viz 5.6.3). Tato metoda neumožňuje žádné zpracování zaznamenávaného videa, není tak možné použít tuto metodu pro záznam v nekonečné smyčce. Ovládání nahrávání videa je prováděno pomocí grafického uživatelského rozhraní (viz Kapitola 7).

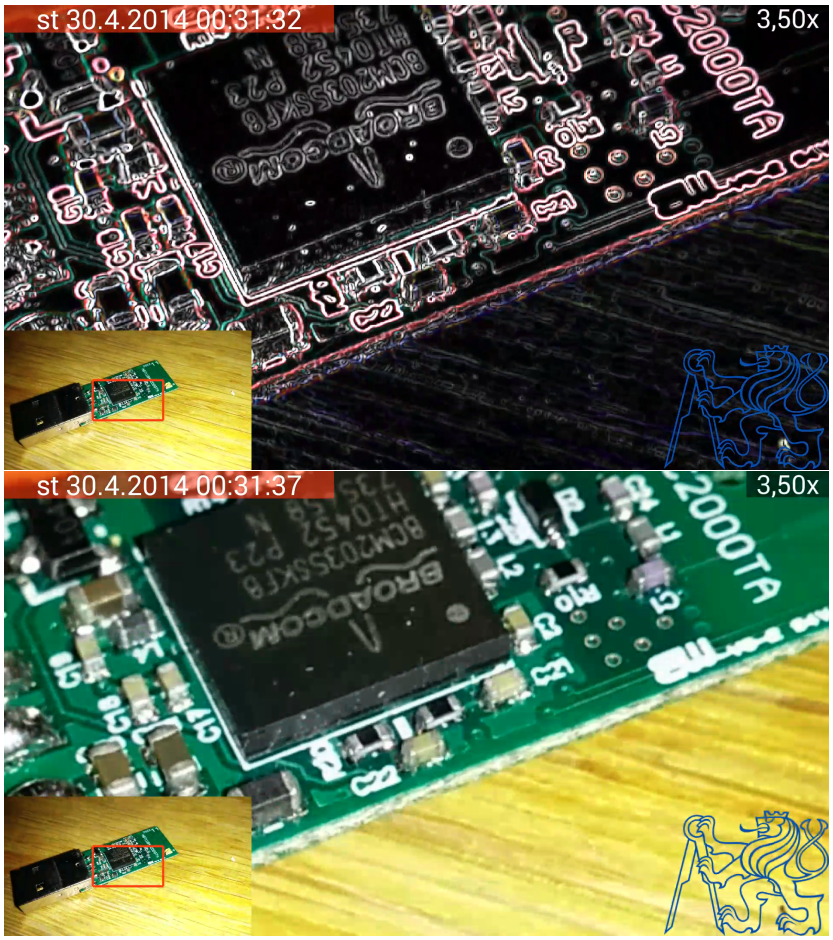
Při nahrávání touto metodou se změní způsob nastavení digitálního zoom na rozsah, který podporuje hardware kamery. Nahrávaný obraz v tu chvíli není zpracováván pomocí OpenGL ES. Zoom se tak aplikuje přímo v hardware kamery a projeví se i v nahraném videu. Rozsah zoom je však dán výrobcem a v případě použitého zařízení (viz 4.3.1) je maximální možný zoom 4×.

5.6.1.2. Textura v OpenGL ES a MediaCodec

Použití nahrávání z OpenGL ES textury pomocí třídy *MediaCodec* je uvedeno v 5.4.2.5. Možnosti nastavení nahrávání v grafickém uživatelském rozhraní jsou popsány v Kapitole 7.4.1 v závislosti na podporovaných video profilech (viz 5.6.3). Tato metoda je založena na vykreslování obrazu z kamery pomocí OpenGL ES, kdy je následně nahrán obraz, který byl grafickou knihovnou zobrazen na displeji. Výhodou této metody je,

že se do videozáznamu ukládá to, co uživatel vidí na displeji, případně za doplnění i dalších úprav obrazu.

Nahrávání videa s obrazovými efekty V případě aplikace Zoomera se obraz nahrává spolu s aplikovaným digitálním zoom v rozsahu 1×-20× nezávisle na modelu zařízení. Ve výsledném záznamu bude též viditelná miniatura obrazu, pokud byla během nahrávání zobrazena. Tyto funkce představují výhody oproti systémovému nahrávání a mohou mít pro uživatele přínos. Video je možné nahrát i s dalšími efekty: detekce hran, hodnota zoom, časová značka, obrázek/logo. Detekce hran byla implementována již v [9]. Protože je detekce hran sama o sobě výpočetně náročná, její záznam do videa si vyžaduje značné systémové prostředky, což způsobuje snížení frekvence vykreslování.



Obrázek 42. Dva snímky z videa pořízené aplikací Zoomera. Znázorňují použití efektů ve videu. Horní obrázek zobrazuje přiblížený snímek z kamery s detekcí hran, dolní obrázek zobrazuje přiblížený snímek z kamery bez úprav. Oba obrázky mají v levém dolním rohu miniaturu obrazu, v levém horním rohu obrazu je časová značka, v pravém honím rohu obrazu je aktuální hodnota zoom a v pravém dolním rohu obrazu je logo ČVUT.

Princip přidání efektů spočívá v uložení příslušných obrazových dat (vykreslený text pro hodnotu zoom a časovou značku nebo obrázek/logo) do OpenGL textury (viz Obrázek 43). Je nutné provést generaci textury pomocí funkce *GLES20.glGenTextures()*[30]. Pro vygenerování dat použijeme třídy *Bitmap*[47] a *Canvas*[48], pomocí které provedeme samotné vytvoření grafiky. Pomocí funkce *GLUtils.texImage2D()*[49] nahrajeme obrazová data do textury. Samotné vykreslení již probíhá podobně jako v případě snímků z

```
1  // Vygenerovani textury
2  int[] textures = new int[1];
3  GLES20.glGenTextures(1, textures, 0);
4  int glTexture = textures[0];
5  // Pripojeni OpenGL textury
6  GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, glTexture);
7  // Nastaveni parametru pro zvetsovani textury
8  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MIN_FILTER,
       GLES20.GL_LINEAR);
9  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_MAG_FILTER,
       GLES20.GL_LINEAR);
10 GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_WRAP_S,
       GLES20.GL_CLAMP_TO_EDGE);
11 GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D, GLES20.GL_TEXTURE_WRAP_T,
       GLES20.GL_CLAMP_TO_EDGE);
12 // Vytvoreni obrazku Bitmap a objektu Canvas, ktery slouzi ke kresleni
13 Bitmap bitmap = Bitmap.createBitmap(width, height,
       Bitmap.Config.ARGB_8888);
14 Canvas canvas = new Canvas(bitmap);
15 // ... vytvoreni pozadovaneho obrazoveho vystupu pomoci canvas.drawXXXX()
16 // Nahrani obrazku do OpenGL textury
17 GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, bitmap, 0);
```

Obrázek 43. Vytvoření OpenGL textury a naplnění jí obrazovými daty z třídy Bitmap.

kamery (viz 5.3.2).

Hodnota zoom Dalším možným obrazovým efektem je zobrazení hodnoty digitálního zoom v pravém horním rohu snímku (viz Obrázek 42). Tento efekt je aplikován pouze na video, vykreslený obraz na displej tento efekt neumožňuje.

Časová značka Do nahrávaného videa je možné vkládat časovou značku. Ta se zobrazuje v levém horním rohu každého snímku (viz Obrázek 42). Časová značka obsahuje datum a hodiny. Tento efekt je aplikován pouze na video, vykreslený obraz na displej tento efekt neumožňuje.

Obrázek/logo Posledním implementovaným efektem je zobrazení loga/obrázku v pravém dolním rohu (viz Obrázek 42). V aplikaci je možné zobrazit pouze logo ČVUT. Použití jiných obrázků metoda nevylučuje, ale pro demonstrační účely použití uvedeného loga postačuje. Tento efekt je aplikován pouze na video, vykreslený obraz na displej tento efekt neumožňuje.

5.6.2. Nahrávání videa ve smyčce

Použití nahrávání ve smyčce je popsáno v Kapitole 5.5. Aplikace Zoomera implementuje tuto funkci pouze ve spojení s nahráváním pomocí OpenGL ES textury a třídy *MediaCodec*. Jiné metody nahrávání neumožňovaly implementaci nahrávání v nekonečné smyčce. Uživatel má možnost nastavit kvalitu nahrávání v grafickém uživatelském rozhraní (viz 7.4.1) v závislosti na podporovaných video profilech (viz 5.6.3). Dále je možné nastavit samotnou délku video smyčky.

V případě, že uživatel spustí nahrávání ve smyčce, je prováděn záznam způsobem, který je popsán v 5.5. Po skončení nahrávání se v závislosti na délce smyčky zobrazí dialog informující uživatele o zpracovávání videa. Dochází k zpracovávání dočasných souborů a vytváření výstupního MP4 video souboru. Délka tohoto zpracování je přímo závislá na velikosti zpracovávaných souborů, protože převážná část zpracování spočívá v kopírování dat z dočasných souborů do výstupního souboru.

Protože je možné pomocí třídy *MediaFormat* nastavit frekvenci vytvoření synchronizačního snímku (viz Obrázek 30), je tato volba přítomná ve video profilech pro metodu nahrávání pomocí OpenGL ES textury a třídy *MediaCodec*. Značena v popisu profilu *(iF:xs)*, kde x je perioda ve vteřinách, se kterou mají být generovány synchronizační snímky. Tato hodnota je důležitá především při použití nahrávání ve smyčce, protože v tomto případě dochází při zpracování videa k ořezávání smyčky na místě, kde je synchronizační snímek. Platí tak, že čím menší perioda výskytu synchronizačního snímku, tím přesněji je možné oříznout výslednou video smyčku.

5.6.3. Video profily

Pro uživatelsky přívětivější nastavení parametrů nahrávání byly vytvořeny video profily, které umožní před začátkem nahrávání nastavit jednu z předdefinovaných konfigurací (viz Kapitola 7.4.1). Byla vytvořena třída *VideoRecorderProfile*, která umožňuje generaci a nastavení takových profilů. Profily byly vytvořeny pro systémové nahrávání a nahrávání pomocí OpenGL ES textury a třídy *MediaCodec*. Důležitou součástí této třídy je kontrola, zda je vytvořená konfigurace validní. Ta se provádí pro každý typ nahrávání jiným způsobem.

5.6.3.1. Systémové nahrávání

Hodnota	**Rozlišení videa**	**Snímkovací f. [fps]**
CamcorderProfile.QUALITY_1080P	1920×1080	30
CamcorderProfile.QUALITY_720P	1280×720	30
CamcorderProfile.QUALITY_480P	720×480	30
CamcorderProfile.QUALITY_QVGA	320×240	30
CamcorderProfile.QUALITY_CIF	352×288	15

Tabulka 3. Přehled systémových video profilů použitých v aplikaci.

Systémové nahrávání používá profily, které jsou definované systémem. Ověření, zda je vybraný profil podporován se provede voláním *CamcorderProfile.get()* s argumentem nesoucím vybranou hodnotu z Tabulky 3. Pokud metoda vrátí instanci *CamcoderProfile*[32], je profil podporován. Pokud vrátí hodnotu *null*, podporován není.

5.6.3.2. Textura v OpenGL ES a MediaCodec

Profily pro video nahrávání byly pro tuto metodu definovány na základě volby několika parametrů. Patří mezi ně rozlišení, datový tok a perioda vytváření synchronizačních snímků. Vytvořené profily zobrazuje Tabulka 4. Nahrávání videa ve smyčce používá stejný základ pro kódování videa, a tak funguje i nastavení video profilu stejným způsobem. Ověření, zda je vybraný profil podporován třídou *MediaCodec* se provede vytvořením instance třídy *MediaFormat* s odpovídajícím nastavením, která se použije pro vytvoření instance třídy *MediaCodec*. Pokud nedojde po zavolání metody *MediaCodec.configure()* k vrácení výjimky, je nahrávací profil podporován. V případě, že se výjimka objeví, profil podporován není.

V tomto případě chybí definice snímkové frekvence. Tu není možné určit, protože je vykreslování a tím i generace nových video snímků závislá na rychlosti s jako jsou jednotlivé snímky vykreslovány a tedy i na výkonu zařízení. Není tak možné zaručit jakoukoli požadovanou snímkovou frekvenci. Je však možné určit s jakou periodou jsou

generovány I-snímky, protože ty jsou závislé na použitém kodeku, který se řídí podle časových značek jednotlivých snímků.

Rozlišení	**Datový tok [bps]**	**I-frame [s]**
3840×2160 *	25000000	5
1920×1080	16000000	5
1920×1080	8000000	1
1920×1080	4500000	10
1280×720	11500000	5
1280×720	6000000	1
1280×720	2500000	10
1024×576	5000000	5
960×540	4500000	5
720×540	4000000	5
704×396	2500000	5
704×396	3500000	1
640×480	3000000	5
640×360	2500000	1
320×240	1000000	5
320×180	400000	5

Tabulka 4. Přehled video profilů pro nahrávání pomocí OpenGL textury a třídy MediaCodec použitých v aplikaci.
** Profil je pouze experimentální. Vstupní obraz z kamery má rozlišení nejvýše 1920×1080.*

6. Hlasové ovládání

Důvodem pro implementaci hlasového ovládání je umožnit použití aplikace Zoomera jako záznamového zařízení v místech, kde je obtížené nebo jakýmkoli způsobem nevýhodné ovládat aplikaci dotykem. Může se jednat o použití v laboratořích, kde by mohlo dojít ke kontaminaci a tak není možné přístroj ovládat ručně. Dalším možným použitím jsou situace, kdy se uživatel musí manuálně věnovat jiné činnosti a nemůže tak ovládat zařízení běžným způsobem pomocí dotyku.

Následující část práce pojednává o teorii týkající se řeči samotné, její digitální reprezentaci a principech a metodách rozpoznávání řeči. Obsahuje vyhodnocení open source knihoven pro rozpoznávání řeči, způsob implementace jejího kontinuálního rozpoznávání, implementaci na systému Android, definici způsobu hlasového ovládání a samotnou implementaci v aplikaci Zoomera.

6.1. Struktura řeči

Řeč je lidská schopnost určená k dorozumívání jedinců a k přenosu informací. Primární podmět ke vzniku řeči dává mozek, který pomocí stahování svalů ovládá lidské hlasivky a za využití okolního vzduchu, se řeč šíří jako zvukové vlny. Na výslednou podobu řeči mají vliv lidská ústa, která ji ovlivňují pohybem čelistí, postavením jazyka a jeho interakcí se zuby. [50]

Řeč se liší v závislosti na fyzických proporcích mluvčího, jazyce, který používá, a na jeho přízvuku. Řeč je tak velmi rozmanitý způsob výměny informací, který má mnoho podob a z jistého pohledu je unikátní u každého mluvčího.

6.1.1. Fonémy

Základní stavební jednotkou řeči je z hlediska zvukového vyjádření foném. Znění fonému je závislé na jazyce, mluvčím, ale také na aktuálním kontextu řeči. Řeč můžeme popsat pomocí teorie pravděpodobnosti tak, že můžeme analyzovat pravděpodobnost, je na sebe konkrétní fonémy navazují. Z fonémů a jejich kombinací jsou tvořena slova. [51, 50]

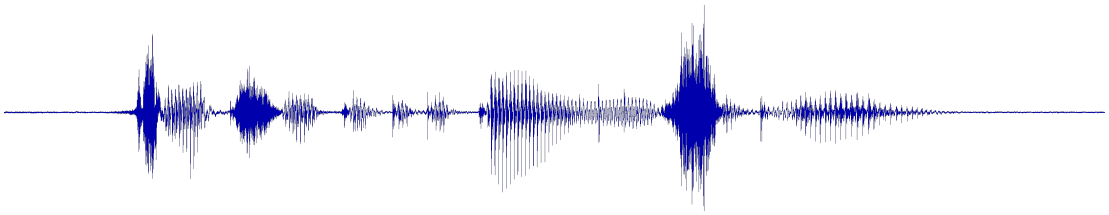
6.1.2. Slova

Slova jsou jazykové elementy, které mají pro člověka určitý význam. Díky nim je možné zavést pravidla pro spojování fonémů s využitím pravděpodobnosti výskytu a jejich propojení. Výpovědi jsou tvořeny pomocí slov a doplňkových zvukových doprovodů, jako je odkašlání nebo jiné zvukové vyjádření, které nemá nic společného s používaným jazykem. [51, 50]

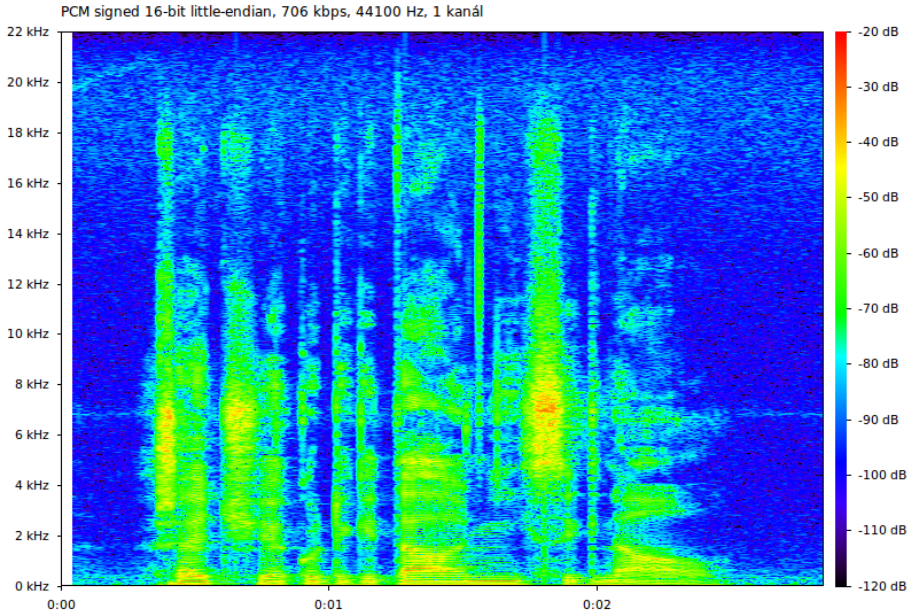
6.1.3. Řeč jako digitální signál

Řeč můžeme uchovávat ve formě digitálního signálu. Ten můžeme definovat jako sérii vzorků, které byly zachyceny s pravidelnou frekvencí. Každý vzorek představuje

jednu hodnotu úrovně zvukové vlny. Příklad grafického zobrazení zvukového záznamu je na Obrázku 44. Zde jsou vidět zaznamenané úrovně zvukového zdroje v závislosti na čase, jedná se o zobrazení v časové oblasti. Pro analýzu zvukového záznamu z hlediska obsažených frekvencí je vhodné zobrazení záznamu v frekvenčním spektru. Příklad je možné pozorovat na Obrázku 45, jedná se o spektrogram. Zobrazuje obsažené frekvence v závislosti na čase. [50]



Obrázek 44. Grafická reprezentace časového průběhu zvukového signálu. Záznam obsahoval řeč: "*Řeč jako digitální signál.*"



Obrázek 45. Spektrogram zvukového záznamu graficky znázorňuje změnu frekvenčního spektra v čase. Záznam obsahoval řeč: "*Řeč jako digitální signál.*"

6.1.3.1. Vzorkovací frekvence

Vzorkovací frekvence je parametr digitálního zvukového záznamu, který udává počet vzorků, který obsahuje jedna vteřina záznamu. V případě rozpoznávání řeči se setkáváme s vzorkovacími frekvencemi 8 a 16 kHz, které jsou pro záznam hlasu dostačující. V oblasti hudby jsou však nároky na kvalitu záznamu vyšší, zde se setkáváme se vzorkovacími frekvencemi 44,1 a 48 kHz. ([50], s. 215)

6.1.3.2. Bitová hloubka vzorku

Bitová hloubka vzorku udává, kolik je přiděleno bitů na uložení jednoho vzorku. Čím vyšší hodnota, tím je možný větší rozsah pro vzorek a tedy i vyšší kvalita záznamu. [50]

6.1.3.3. Pulzně kódová modulace

Pulzně kódová modulace je způsob uložení digitálního zvukového záznamu. Je specifikována vzorkovací frekvencí (viz 6.1.3.1) a počtem bitů na vzorek (viz 6.1.3.2). Digitální signál je uložen po pravidelně nahraných vzorcích získaných kvantizací analogového signálu na nejbližší hodnotu v rozsahu. Jedná se o posloupnost celých čísel, kde každé představuje úroveň jednoho vzorku. [50]

6.2. Rozpoznávání řeči

Možnost provést rozpoznávání řeči vyžaduje porozumění struktury řeči a tomu jak vzniká, reprezentaci řeči jako digitálního signálu a metodám k nalezení popisu zvukového vyjádření řeči, které umožní rozpoznávání za pomocí modelů konkrétního jazyka.

6.2.1. Extrakce a popis příznaků

Máme-li digitální audio záznam a chceme provést rozpoznávání řeči, je nutné v něm nejprve detekovat úseky, kde se nachází řeč a odstranit úseky, kde se nachází pouze ticho nebo zvuky okolí. Používá se k tomu detektor, který v záznamu hledá ukončení výpovědi, za pomoci prahování, které dynamicky reaguje na průběh signálu v čase. Po nalezení všech ukončení se záznam rozdělí na jednotlivé úseky, na kterých bude detekce spuštěna odděleně. ([50], s. 421)

Ve chvíli, kdy máme úryvek zvukové nahrávky a chceme v ní detekovat fonémy a celá slova, je nutné, aby bylo možné najít v signálu oblasti, které jsou důležité z hlediska odlišení jednotlivých fonémů. Tyto oblasti je nutné popsat co nejpřesněji, ale zároveň vektorem v nízké dimenzi, abychom mohli takový popis srovnávat s předlohou v rozumném výpočetním čase. Proces nalezení takových oblastí se nazývá extrakce příznaků. [51, 50]

Pro samotnou extrakci příznaků z audio signálu se používají melovské kepstrální koeficienty (mel-frequency cepstral coefficients - MFCC). Jedná se o příznaky frekvenčního spektra, které je méně náchylné na ovlivnění šumem a zkreslením než v případě hledání příznaků v časové oblasti. Metoda hledání MFCC spočívá v aplikaci Fourierovy transformace na audio signál. Fourierova transformace se neprovádí najednou na celém signálu, ale jen v jeho výřezu (plovoucí okno). Následuje použití mel filtrové banky. Jedná se o soubor triangulárních signálových filtrů, kde každý přísluší určité frekvenci, pomocí nichž se vypočítá průměrné spektrum okolo příslušných frekvencí. Na filtrované frekvenční spektrum je aplikována diskrétní kosinová transformace, díky které získáme melovské kepstrální koeficienty. Pro účely rozpoznávání řeči se nejčastěji používá prvních 13 příznaků. [52], ([50], s. 314)

Získáme tak vyjádření fragmentu audio signálu v nižší dimenzi o velikosti 13, jehož porovnání s jinými takto vyjádřenými fragmenty je robustnější a rychlejší než v případě porovnávání fragmentů signálu v časové oblasti. Popsaný proces je nevratný, můžeme hovořit o aproximaci konkrétního fragmentu audio signálu. ([50], s. 423)

6.2.2. Skrytý Markovův model

Skrytý Markovův model (Hidden Markov model - HMM) je založen na Markovově řetězci. To je pravděpodobnostní proces modelující náhodné procesy, který je definován stavy a orientovanými hranami mezi nimi. Neexistují paralelní hrany. Každá hrana je ohodnocena číslem od 0 do 1, které udává pravděpodobnost že dojde k přechodu po této

hraně do cílového stavu. Pravděpodobnosti přechodů závisejí pouze na stavech odkud vycházejí a nejsou ovlivněny stavy, které byly dříve navštíveny. Součet pravděpodobnostních ohodnocení hran vycházejících z jednoho stavu je vždy roven 1. [50]

Markovův řetězec je možné popsat maticí přechodů A o velikosti N×N, kde N je počet stavů. Prvek a_{ij} matice A obsahuje pravděpodobnostní ohodnocení hrany ze stavu i do stavu j. Dále definujeme vektor počátečních pravděpodobností stavů π, kde π_i je rovno pravděpodobností, že počátečním stavem je i-tý stav. Pravděpodobnost sekvence přechodů vypočteme vynásobením příslušné pravděpodobnosti počátečního stavu pravděpodobnostmi všech provedených přechodů. [50]

Skrytý Markovův model (HMM) poté definujeme jako Markovův řetězec, který obsahuje skryté stavy. Markovův řetězec můžeme pozorovat na základě stavu, ve kterém se nachází a na základě výstupu obsahujícího sekvenci dříve navštívených stavů. Markovův řetězec je tak parametrizován pouze pravděpodobnostmi přechodů. Oproti tomu HMM u každého stavu definuje vektor udávající pravděpodobnosti s jakými se stav jeví pozorovateli. Pozorovatel tak nemá již jistotu, že stav, ve kterém se dá HMM pozorovat, je stavem, ve kterém se HMM opravdu nachází. [50]

6.2.3. Řečové modely

Řečové modely slouží k popisu mluveného slova a jsou používány v průběhu rozpoznávání řeči. Pro popis řeči definujeme tři modely: akustický, fonetický a jazykový. [51]

6.2.3.1. Akustický model

Akustický model popisuje akustické vlastnosti jednotlivých fonémů, propojuje vektory příznaků a fonémy. Tento model se dělí na dva typy podle uvažování kontextu. Prvním je typ, který je na kontextu výskytu fonému nezávislý, a tak obsahuje pouze nejpravděpodobnější příznakové vektory pro každý foném. Druhým typem je model závislý na kontextu výskytu fonému. Jedná se o model, který obsahuje statistické reprezentace, které jsou přidružené k jednotlivým fonémům. Tyto statistické reprezentace jsou tvořeny Skrytými Markovovými modely (viz 6.2.2), tedy každý foném je reprezentován jedním HMM. [51, 50]

6.2.3.2. Fonetický model

Fonetický model slouží k propojení slov a příslušných fonémů, ze kterých se slovo skládá. Komplexita tohoto modelu může značně narůstat, protože existují slova, která je možné vyslovovat v několika možných variantách. [51]

6.2.3.3. Jazykový model

Jazykový model popisuje použitý jazyk z hlediska skladby a spojování jednotlivých slov. Model tak definuje, která slova mají zvýšenou pravděpodobnost výskytu v závislosti na historii již rozpoznaných slov. Tím je umožněno zmenšit prohledávaný prostor slov při rozpoznávání jen na ta, která mají největší pravděpodobnost, že budou navazovat na předešlá slova. [51]

Jazykový model je vytvořen pro konkrétní jazyk a je definován gramatikou. Ta udává všechny povolené struktury, které mohou v jazyce vzniknout. Tento způsob je však složitý pro použití v jazycích s bohatou strukturou a slovní zásobou. Z toho důvodu

existuje i druhý způsob definice jazykového modelu. Ten je popsán jako pravděpodobnostní model, který definuje pravděpodobnost výskytu pro různé sekvence slov. [50]

6.2.4. Proces rozpoznávání

Rozpoznávání řeči je založeno na rozpoznávání jednotlivých slov, ze kterých se řeč skládá. Vstupem je zvukový záznam, ve kterém je provedena detekce pomlk, dle nich je záznam rozdělen na jednotlivé výpovědi. Následně se ve zvukových fragmentech provede nalezení příznakových vektorů, které jsou použity k nalezení příslušných fonémů pomocí akustického modelu. [51]

Za použití akustického a jazykového modelu je vytvořen vyhledávací graf tvořen Skrytými Markovovými modely fonémů, který představuje jednotlivá slova jazyka. Fonémy nalezené ve zvukovém fragmentu jsou použity při prohledávání prostoru grafu. Hledá se nejlepší shoda taková, že pravděpodobnost, že jednotlivé fonémy byly generovány jako jedno konkrétní slovo, je největší. [51]

6.2.4.1. Metriky prohledávání

Protože prohledávání prostoru možných slov je výpočetně složitý problém, existují metriky, které umožňují definovat kvalitu výsledků a umožnit optimalizaci prohledávání.

Chybovost slov (Word error rate) Rozpoznaný text má N slov. I udává počet slov, která byla do rozpoznaného textu oproti původnímu textu vložena, D je počet smazaných slov z rozpoznaného textu a S je počet slov, která byla oproti původnímu textu změněna. [51]

Chybovost slov $WER = (I + D + S)/N$.

Přesnost rozpoznání Rozpoznaný text má N slov. I udává počet slov, která byla do rozpoznaného textu oproti původnímu textu vložena, D je počet smazaných slov z rozpoznaného textu a S je počet slov, která byla oproti původnímu textu změněna. [51]

Přesnost rozpoznání $A = (N - D - S)/N$.

6.3. Knihovny pro rozpoznávání řeči

6.3.1. Open source knihovny

Pro účely práce byl nalezen a vyhodnocen vzorek open source knihoven pro rozpoznávání řeči dostupný na internetu. Z tohoto vzorku byla vybrána jedna knihovna, následně použita v implementaci hlasového ovládání. Tato knihovna musí umožňovat použití na platformě Android, musí obsahovat anglický jazykový model, musí být možné upravovat jazykový a akustický model pro přizpůsobení rozpoznávání řeči a je vyžadována podpora kontinuálního rozpoznávání.

6.3.1.1. CMU Sphinx

Projekt Carnegie Mellon University nazvaný CMU Sphinx [53] je systém určený pro rozpoznávání řeči. Je vybaven několika nástroji pro různé typy použití. Balík Sphinx4 umožňuje rozpoznávání řeči s širokou možností nastavení a úprav jazykových a akustických modelů. Je kompletně napsán v jazyce Java. Jednodušší variantou je nástroj Pocketsphinx určený pro rozpoznávání řeči na vestavěných systémech. Je napsán v jazyce C, je dostupný též pro systém Android a iOS a dodáván s licencí odvozenou od BSD a Apache.

6.3.1.2. Julius

Projekt Julius [54] je open source knihovna pro rozpoznávání řeči napsaná v jazyce C. Systém umožňuje rozpoznávání řeči v reálném čase a širokými možnostmi nastavení. Je dostupný pro Windows, Linux, Mac OS X a Solaris pod licencí odvozenou z BSD. Verze pro mobilní platformy není k dispozici.

6.3.1.3. OpenEars

OpenEars [55] je knihovna pro rozpoznávání řeči a syntézu hlasu na mobilních zařízeních se systémem iOS. Pro rozpoznávání řeči využívá knihovnu CMU Pocketsphinx. Část projektu přebírá licenci typu BSD od CMU Pocketsphinx, zbylá část je pod licencí Politepix Public License verze 1.0. Verze pro systém Android není k dispozici.

6.3.1.4. Kaldi

Kaldi [56] je soubor nástrojů pro rozpoznávání řeči napsaný v jazyce C++ a distribuovaný pod licencí Apache License v2.0. Je převážně navržen pro použití výzkumníky v oblasti rozpoznávání řeči. Je distribuován pod licencí Apache a dostupný pro Windows a Unixové systémy. Verze pro mobilní platformy není k dispozici.

6.3.1.5. Android SpeechRecognizer

SpeechRecognizer [57] je součást open source systému Android umožňující rozpoznávání řeči. Nástroj byl z počátku navržen tak, aby veškeré rozpoznávání probíhalo na serverech společnosti Google, Inc. Později byla přidána možnost provádět rozpoznávání přímo na mobilních zařízeních se systémem Android bez potřeby připojení k internetu. Nedoporučuje se používat tento nástroj pro kontinuální rozpoznávání. Nástroj neumožňuje úpravu jazykových ani akustických modelů rozpoznávání.

6.3.2. Použité licence

6.3.2.1. Apache License

Apache License je licence svobodného software společnosti Apache Software Foundation. Vyžaduje uvedení informací o licenčních podmínkách a zřeknutí se odpovědnosti. Dovoluje volné šíření a úpravu díla. [58]

6.3.2.2. BSD

BSD je licence svobodného software, která umožňuje volné šíření díla. Je vyžadováno uvedení informace o licenčních podmínkách, autora a zřeknutím se odpovědnosti. [59]

6.3.2.3. Politepix Public License verze 1.0

Politepix Public License verze 1.0 je licence vytvořena přímo pro OpenEars projekt. Ten obsahuje několik knihoven třetích stran (převážně CMU knihovny), které mají svoji licenci. Licence umožňuje provádět modifikace, linkovat, kompilovat, připojovat knihovnu a vytvářet s ní i komerční aplikace. Umožňuje distribuci takových aplikací v binární podobě. Vyžaduje uvedení autora knihovny v aplikaci.

6.3.3. Vybraná knihovna - CMU Sphinx

Z nalezeného vzorku knihoven pro rozpoznávání řeči byla vybrána jedna, která splňuje všechny výše zmíněné požadavky. Jediná knihovna z nalezeného vzorku, která těmto požadavkům odpovídala byla **CMU Sphinx**.

Knihovna CMU Sphinx je open source projektem Carnegie Mellon University, fungující jako rozsáhlý systém pro rozpoznávání řeči s nástroji pro vytváření a editování vlastních jazykových a akustických modelů řeči.

6.3.3.1. Součásti knihovny

Knihovna CMU Sphinx se skládá z jednotlivých součástí/projektů. Každý z nich pokrývá určitou funkčnost celého systému od samotného nastavitelného rozpoznávání řeči, přes jednodušší rozpoznávání řeči na vestavěných zařízeních, až po nástroje pro vytváření a úpravu jazykových modelů a trénování akustických modelů. [60]

Sphinx4 je knihovna, která slouží k samotnému rozpoznávání řeči, umožňuje detailní nastavení rozpoznávání, použití vlastních jazykových a akustických modelů. Je kompletně napsaná v jazyce Java. Byla postavena na základech knihovny Sphinx3.

Sphinx3 je knihovna určená k rozpoznávání řeči. Je napsaná v jazyce C.

Pocketsphinx jednodušší verze knihovny pro rozpoznávání řeči pro vestavěná zařízení. Je napsaná v jazyce C. Je použitelná v systémech Google Android a iOS.

Sphinxbase pomocná knihovna pro Pocketsphinx.

CMUclmtk obsahuje soubor nástrojů pro práci s jazykovými modely.

Sphinxtrain obsahuje soubor nástrojů pro trénování akustických modelů.

6.4. Rozpoznávání řeči v aplikaci Zoomera

Cílem bylo do aplikace Zoomera implementovat modul rozpoznávání řeči tak, aby bylo možné ovládat všechny funkce aplikace hlasem. Dotyk či jiný způsob uživatelského vstupu kromě hlasového vstupu je potřeba pouze na spuštění a zastavení tohoto modulu. Pro samotné rozpoznávání řeči byla využita knihovna CMU Sphinx.

6.4.1. Záznam zvuku

Pro možnost rozpoznání řeči je nejdříve nutné zaznamenat uživatelský vstup pomocí mikrofonu připojeného k zařízení. Data reprezentující tento zvukový vstup mohou být následně poskytnuta knihovně pro rozpoznávání řeči. Jako možné periferní zařízení pro záznam zvuku byl uvažován mikrofon vestavěný v mobilním zařízení, připojitelný externí mikrofon a Bluetooth náhlavní souprava nebo sluchátka.

6.4.1.1. Záznam zvuku na platformě Android

Nahrávání zvuku v systému Android je možné provádět dvěma způsoby. Prvním je použití třídy *MediaRecorder*[31]. Tento způsob však není v práci použit, umožňuje nahrávání pouze do souboru a byl by pro použití kontinuálního rozpoznávání nevhodný. Nebylo by v tomto případě efektivní zaznamenávat zvuk do souboru a znovu ho ze souboru číst za účelem rozpoznávání. Dalším důvodem je skutečnost, že třída *MediaRecorder* neumožňuje nahrávání do formátu WAVE resp. PCM (viz 6.1.3.3), který je vyžadován knihovnou CMU Sphinx pro rozpoznávání.

AudioRecord Druhým možným způsobem nahrávání je využití třídy *AudioRecord* [61]. Tato třída disponuje metodou *read*, která zapíše předem definovaný fragment zvukového záznamu do pole byte, short nebo do objektu ByteBuffer. Tímto způsobem je zvuk dostupný přímo v operační paměti zařízení a je možné ho jakkoli zpracovávat.

Základní použití této metody je ilustrováno na Obrázku 46. Zde je nakonfigurována třída *AudioRecord* na nahrávání audia s 16000 Hz vzorkovací frekvencí (viz 6.1.3.1), jedním zvukovým kanálem a ve formátu PCM s 16 bity na vzorek (viz 6.1.3.2). Jako zdroj zvuku je vybrán mikrofon zařízení. Tato konfigurace je vyžadována pro použití audia při rozpoznávání řeči knihovnou Pocketsphinx [62].

Spuštění nahrávání se provádí zavoláním metody *startRecording()* ve třídě *AudioRecord*. Od této chvíle systém nahrává zvuk ze zvoleného vstupu s nastavenou konfigurací. Pokud chceme získat nahrávaný zvuk, zavoláme metodu *read()* s parametrem *buffer*, který obsahuje ukazatel na pole, do kterého chceme fragment nahraného audia uložit. Ukončení nahrávání provedeme zavoláním metody *stop()* třídy *AudioRecorder* následovanou metodou *release()*, která uvolní všechny prostředky použité při nahrávání.

Popsaná ukázka kódu demonstruje jen triviální použití třídy *AudioRecord* pro nahrávání. Protože se předpokládá, že zpracování nahraného audia vyžaduje větší množství výpočetního výkonu, používá se vícevláknového přístupu. V případě nahrávání zvuku pro rozpoznávání řeči v aplikaci Zoomera je metoda *read()* volána v cyklu ze samostatného vlákna. Všechny audio fragmenty v short poli se ukládají do thread-safe fronty *LinkedBlockingQueue*. Ta umožňuje použití návrhu typu producent-konzument. V tomto případě je producentem vlákno, které volá metodu *AudioRecord.read()* a ukládá audio fragmenty, konzumentem je knihovna pro rozpoznávání řeči, která z fronty postupně odebírá audio fragmenty a zpracovává je. Tento způsob je použitý v aplikaci Zoomera a je implementován spolu s frontou v třídě *AudioRecorder*.

```
1   int BUFFER_SIZE = 2048;
2   int audioSource  = MediaRecorder.AudioSource.MIC;
3   int sampleRateInHz = 16000;
4   int channelConfig = AudioFormat.CHANNEL_IN_MONO;
5   int audioFormat = AudioFormat.ENCODING_PCM_16BIT;
6   int bufferSizeInBytes = AudioRecord.getMinBufferSize(sampleRate,
7                                                        channels,
8                                                        audioFormat);
9
10  AudioRecord audioRecord;
11  audioRecord = new AudioRecord(audioSource, sampleRateInHz,
12                                channelConfig, audioFormat,
13                                bufferSizeInBytes);
14
15  audioRecord.startRecording();
16  short[] buffer = new short[BUFFER_SIZE];
17  int result = this.audioRecord.read(buffer, 0, BUFFER_SIZE);
18  if (result == AudioRecord.ERROR_INVALID_OPERATION ||
19      result == AudioRecord.ERROR_BAD_VALUE) {
20     // CHYBA
21  }
22
23  audioRecord.stop();
24  audioRecord.release();
```

Obrázek 46. Základní použití třídy AudioRecord pro záznam zvuku z mikrofonu zařízení do pole short elementů. Parametry nahrávaného audia jsou 16000 Hz vzorkovací frekvence, mono nastavení zvukových kanálů a formát PCM s 16 bity na vzorek.

6.4.1.2. Vestavěný mikrofon

Prvním uvedeným a nejčastěji používaným zvukovým vstupem na mobilním zařízení je vestavěný mikrofon zařízení. Tento mikrofon se zpravidla používá při telefonních hovorech. Je to výchozí mikrofon, který zařízení má. Protože se používá pro záznam hlasu uživatele, který drží zařízení v blízkosti své hlavy, a tedy i mikrofon je ve velké blízkosti úst, předpokládá se, že jeho rozlišovací schopnosti budou na vzdálenost několika metrů špatné.

Pro použití tohoto mikrofonu pro nahrávání pomocí třídy *AudioRecord* je nutné definovat jako vstup hodnotu *MediaRecorder.AudioSource.DEFAULT* nebo *MediaRecorder.AudioSource.MIC* [63].

6.4.1.3. Externí mikrofon

U mobilních zařízení se dá předpokládat přítomnost 3,5 mm jack konektoru, který umožňuje připojení sluchátek s mikrofonem. Takový externí mikrofon se v případě jeho zapojení do zařízení stane hlavním mikrofonem. U mikrofonu se také jako u vestavěného předpokládá, že se nachází v blízkosti úst uživatele. Výhodou může však být to, že délka přívodního vodiče umožní uživateli volnější pohyb, a i tak bude mít stále externí mikrofon ve stejné vzdálenosti od úst.

Pro použití tohoto mikrofonu pro nahrávání pomocí třídy *AudioRecord* platí stejné nastavení jako u vestavěného mikrofonu. Je nutné externí mikrofon připojit a definovat jako vstup hodnotu *MediaRecorder.AudioSource.DEFAULT* nebo *MediaRecorder.AudioSource.MIC* [63].

6.4.1.4. Přenos zvuku přes Bluetooth

Poslední uvažovanou možností zvukového vstupu je Bluetooth headset nebo Bluetooth sluchátka. Toto řešení využívá bezdrátové komunikace k výměně dat (v tomto

případě audia) mezi dvěma zařízeními. Bluetooth sluchátka bývají vybavena reproduktory, mikrofonem a ovládacími tlačítky (viz Obrázek 47).

Obrázek 47. Bluetooth sluchátka Sony DR-BT21G [64]

Implementace nahrávání audia přes Bluetooth je však složitější než v případě vestavěného a externího mikrofonu. Obnáší navázání Bluetooth spojení a vytvoření audio tunelu mezi zařízením a Bluetooth sluchátky.

Pro navázaní Bluetooth spojení využijeme systémovou třídu *AudioManager* (viz ukázka zdrojového kódu na Obrázku 48) [65]. Nastavíme-li v *AudioManager.setMode()* hodnotu *AudioManager.STREAM_MUSIC*, budeme mít také možnost přehrávat audio ve sluchátkách. To je v aplikaci použito pro přehrání výstupu hlasového syntetizátoru, který čte právě prováděný hlasový příkaz. Abychom se ujistili, že mikrofon není vypnutý, zavoláme příkaz *AudioManager.setMicrophoneMute(false)*. Protože výsledek navázání spojení není možné získat synchronně se spuštěním Bluetooth připojení, použije se objekt třídy rozšiřující *BroadcastReciever*. Metoda *onRecieve()* tohoto objektu bude zavolána v případě, že dojde ke změně na stavu Bluetooth spojení. Spojovaní spustíme metodou *startBluetoothSco()*.

Pokud existuje nějaké spárované multimediální Bluetooth zařízení, které je v danou chvíli v dosahu, proběhne s tímto zařízením navázání spojení. V případě, že takové zařízení neexistuje, ke spojení nedojde, výchozím mikrofonem zůstává poslední použitý.

Když chceme ukončit práci s Bluetooth, obnovíme výchozí mód pomocí metody *AudioManager.setMode()* hodnotu *AudioManager.MODE_NORMAL*. Následně ukončíme Bluetooth spojení pomocí *stopBluetoothSco()* a odstraníme záznam o instanci objektu *BroadcastReciever*. Aktivním zvukovým vstupem se stane výchozí mikrofon.

Pro větší přínos pro uživatele je metoda přenosu zvuku přes Bluetooth v aplikaci Zoomera doplněna o hlasový syntetizátor, který vytváří řečovou reprodukci rozpoznaných příkazů hlasového ovládání. Zpětná vazba pro uživatele není tak jen optická, ale i zvuková. Protože je hlas syntetizován jen v případě použití Bluetooth sluchátek, nemůže docházet ke kolizi hlasového výstupu a hlasového ovládání. Pro vytvoření syntézy hlasu byla použita systémová třída *TextToSpeech*[66].

6.4.2. Definice způsobu ovládání hlasem

Vybraná knihovna pro rozpoznávání řeči nám dává funkci pro převedení zaznamenaného zvuku na text dle modelu jazyka definovaného v knihovně. Pokud ale chceme ovládat aplikaci hlasovým vstupem, je nutné definovat způsob, jak bude textový výstup z knihovny pro rozpoznávání řeči zpracován a jak má uživatelský vstup vypadat.

```
AudioManager audioManager;
audioManager = (AudioManager)
    this.activity.getSystemService(Context.AUDIO_SERVICE);

audioManager.setMode(AudioManager.STREAM_MUSIC);
audioManager.setMicrophoneMute(false);

BluetoothBroadcastReciever bluetoothBroadcastReciever = new
    BluetoothBroadcastReciever();
this.activity.registerReceiver(bluetoothBroadcastReciever,
          new IntentFilter(AudioManager.ACTION_SCO_AUDIO_STATE_UPDATED));

audioManager.startBluetoothSco();

// ...

audioManager.setMode(AudioManager.MODE_NORMAL);
audioManager.stopBluetoothSco();
activity.unregisterReceiver(bluetoothBroadcastReciever);
```

RemoteControlReceiver.java

```
public class BluetoothBroadcastReciever extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      if (AudioManager.ACTION_SCO_AUDIO_STATE_UPDATED
          .equals(intent.getAction())) {
         int state =
             intent.getIntExtra(AudioManager.EXTRA_SCO_AUDIO_STATE, -1);
         if (AudioManager.SCO_AUDIO_STATE_CONNECTED == state) {
            // Kod, ktery se vykona pri navazani spojeni
         }
         if (AudioManager.SCO_AUDIO_STATE_DISCONNECTED == state) {
            // Kod, ktery se vykona pri odpojeni
         }
         if (AudioManager.SCO_AUDIO_STATE_ERROR == state) {
            // Kod, ktery se vykona v pripade chyby
         }
      }
   }
}
```

Obrázek 48. Ukázka zdrojového kódu, který naváže Bluetooth spojení mezi zařízením se systémem Android a Bluetooth sluchátky za účelem bezdrátového přenosu zvuku z mikrofonu sluchátek do zařízení. Využívá se systémové třídy *AudioManager* a třídy *BluetoothBroadcastReciever* rozšiřující systémovou třídu *BroadcastReciever*.

6.4.2.1. Princip ovládání

Chceme zaručit, aby se uživatel nemusel zařízení s aktivním hlasovým ovládáním vůbec dotýkat a jediným vstupem byl jen ten hlasový. Aplikace tak musí kontinuálně poslouchat hlasový vstup a analyzovat, kdy uživatel zadává hlasové příkazy. Rozpoznávání bude navrženo pro anglický jazyk.

Aby bylo jasně definováno, že uživatel chce zadat příkaz, bude použito aktivační slovo, které bude uživatelem vysloveno před každým příkazem. Vstupy tak bude možné filtrovat, protože jen to, co následuje po aktivačním slově, může být platný příkaz.

Aktivační slovo Pro tyto účely musí být definováno takové aktivační slovo, které se nevyskytuje v žádném příkazu, a zároveň takové, které bude jednoduché na výslovnost a zapamatování. Pro aplikaci Zoomera bylo určeno anglické slovo "my"(moje, můj).

Dále musí být pro každou funkci navržen jednoduchý příkaz. Tímto příkazem v kombinaci s aktivačním slovem je možné spustit danou funkci. Příkazy musí být jednoznačné

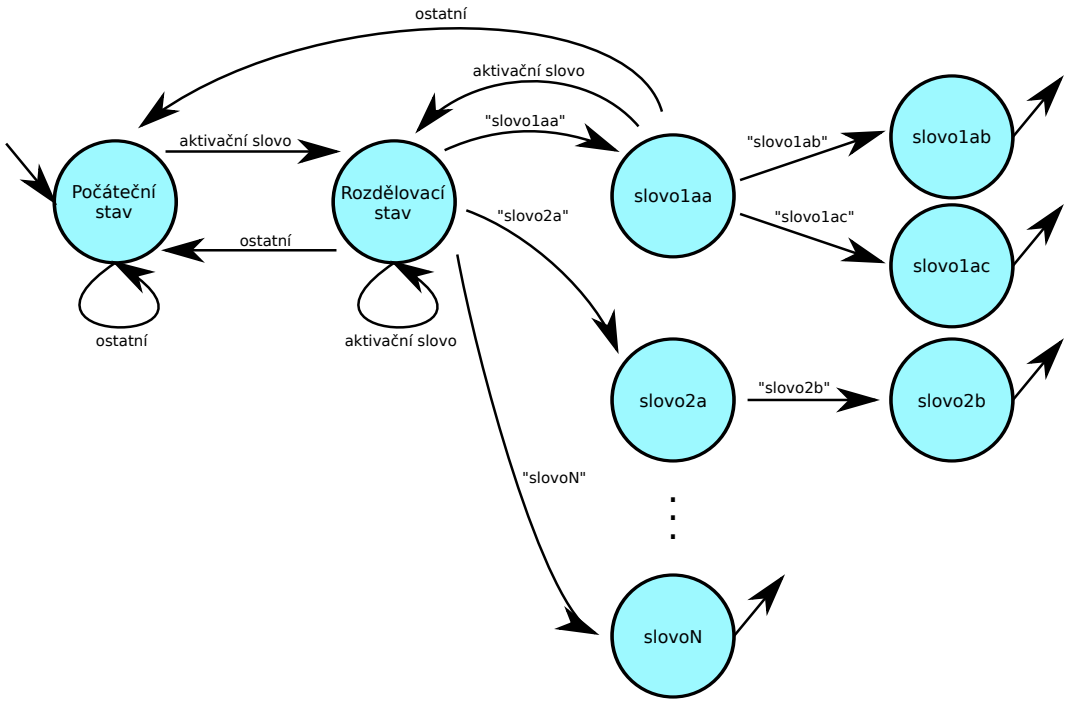
a měly by popisovat funkci, kterou provádějí, aby byly pro uživatele zapamatovatelné. Pokud je to možné, tak by se měly příkazy mezi sebou co nejvíce lišit, aby nedocházelo k jejich záměně při rozpoznávání.

6.4.2.2. Stavový automat

Výstup z knihovny pro rozpoznávání řeči je textový řetězec. Pro zachování plynulosti hlasového ovládání je nutné v tomto textovém řetězci co nejrychleji nalézt aktivační slovo a zadaný příkaz. Protože knihovna rozpoznává řeč kontinuálně, může se stát, že ať už vlivem chyby rozpoznávání nebo zaznamenané řeči, která nesouvisí s ovládáním, textový řetězec obsahuje mnoho rozpoznaných slov, které s ovládáním nesouvisí. Mezi těmito všemi slovy musí být nalezen zadaný příkaz. Pro tyto účely je použit stavový automat, který na základě vstupu obsahujícího všechny slova z textového řetězce, určí příkazy, které se v řetězci nacházejí.

Vstupy automatu jsou všechna slova, která obsahuje slovník jazykového modelu používaného při rozpoznávání řeči.

Stavy automatu jsou generovány z aktivačního slova a ze slov v jednotlivých příkazech. Automat vždy obsahuje počáteční stav, v kterém se nachází ve výchozím nastavení automatu. V koncových stavech se automat nachází v případě, že vstupní řetězec, který je automatem validován, obsahuje platný příkaz.



Obrázek 49. Obecný validační stavový automat. Obsahuje počáteční stav, rozdělovací stav a stavy jednotlivých slov. Ze všech stavů vede hrana do rozdělovacího stavu označená vstupem aktivační slovo a do počátečního stavu označená vstupy, pro které nejsou v daném stavu definovány jiné přechody. Na obrázku jsou tyto hrany pro přehlednost zobrazeny jen u stavu *slovo1aa*. Automat se v tomto případě dostane do koncového stavu, pokud textový řetězec na vstupu obsahuje podřetězec "*aktivační slovo* slovo1aa slovo1ab", "*aktivační slovo* slovo1aa slovo1ac", "*aktivační slovo* slovo2a slovo2b", "*aktivační slovo* slovoN", atd.

Přechodová funkce automatu definuje hrany, které vedou z jednotlivých stavů, a do jakého stavu automat přechází na základě vstupu.

Z počátečního stavu se automat dostane do rozdělovacího stavu v případě, že se na vstupu nachází aktivační slovo. V ostatních případech zůstává v počátečním stavu.

Rozdělovací stav vyžaduje, aby posledním vstupem bylo aktivační slovo. Pokud je na vstupu znovu aktivační slovo, automatu zůstává v rozdělovacím stavu. Z rozdělovacího stavu vedou hrany do stavů, které reprezentují první slovo validního příkazu. V ostatních případech vstupů se automat vrací do počátečního stavu. Stavy, které reprezentují první slovo validního příkazu mohou být i koncovými stavy, v případě, že je příkaz jednoslovný. V ostatních případech pokračuje větvení stavů i do dalších úrovní v závislosti na definovaných validních příkazech.

Ze všech stavů vede zároveň hrana do rozdělovacího stavu, když je na vstupu aktivační slovo. Pokud se v jakémkoli stavu vyskytne na vstupu slovo, pro které není hrana do žádného jiného stavu, automat se vrací do počátečního stavu.

Konstrukce automatu Automat je vytvořen na základě seznamu všech možných příkazů hlasového ovládání. Seznam obsahuje aktivační slovo spolu s danými příkazy. Na základě výše definované přechodové funkce je rekurzivně vytvořen automat, který přijímá všechny příkazy a je schopen přijmout jakýkoli vstup, který se nachází ve slovníku jazykového modelu pro rozpoznávání řeči. Na Obrázku 49 je graficky znázorněn obecný automat odpovídající výše uvedeným podmínkám.

6.4.2.3. Definované příkazy

Pro hlasové ovládání aplikace Zoomera byly definovány následující příkazy. V seznamu jsou uvedeny všechny příkazy spolu s aktivačním slovem "my"a vysvětlením. V závislosti na výsledcích testování byly některé příkazy upraveny (viz 8.2.5).

MY ZOOM INCREASE Zvýší zoom o jeden krok. V aplikaci je krok 0.25×.
MY ZOOM DECREASE Sníží zoom o jeden krok. V aplikaci je krok 0.25×.
MY MINIMAL ZOOM Nastaví minimální zoom. V aplikaci 1.00×.
MY MEDIUM ZOOM Nastaví střední zoom. V aplikaci 10.00×.
MY MAXIMAL ZOOM Nastaví maximální zoom. V aplikaci 20.00×.
MY LIGHT ON Rozsvítí LED.
MY DARK Zhasne LED.
MY ENABLE AUTOFOCUS Povolí kontinuální ostření.
MY DISABLE AUTOFOCUS Zakáže kontinuální ostření.
MY FOCUS Spustí ostření.
MY SHOW MINIATURE Zobrazí miniaturu.
MY HIDE MINIATURE Skryje miniaturu.
MY TAKE PHOTO Vyfotografuje snímek.
MY SAVE PHOTO Uloží vyfotografovaný snímek.
MY RETURN Spustí znovu živý náhled po fotografování.
MY START RECORDING Spustí nahrávání videa.
MY STOP RECORDING Ukončí nahrávání videa.
MY HIDE CONTROLS Skryje ovládací prvky.
MY SHOW CONTROLS Zobrazí ovládací prvky.

6.4.3. Použití a nastavení knihovny CMU Sphinx - Pocketsphinx

Pro rozpoznávání řeči na mobilním zařízení se systémem Android byla použita knihovna Pocketsphinx z projektu CMU Sphinx. Je napsaná v jazyce C a poskytuje takovou rychlost rozpoznávání, která dovoluje použití na vestavěných zařízeních.

6.4.3.1. Požadavky pro systém Android

Pocketsphinx vyžaduje pro začlenění do Android aplikace použití několika technologií. První jsou základní vývojářské nástroje Android SDK, dále nástroje pro nativní vývoj Android NDK. Předpokládá se použití vývojového prostředí Eclipse. Dále je vyžadován Apache Ant, Java Development Kit a Simplified Wrapper and Interface Generator (SWIG) pro generování Java interface umožňujícího ovládání knihovny Pocketsphinx. Pro fungování knihovny je nezbytná instalace knihovny Sphinxbase, která zajišťuje základní funkčnost knihovny Pocketsphinx. Podrobný návod na zprovoznění spolu s ukázkovým projektem, kterým je rozpoznávání řeči v aplikaci Zoomera inspirováno, se nachází na [67, 68].

6.4.3.2. Jazykový a akustický model v CMU Sphinx

Jazykový model Pro vytvoření jednoduchého jazykového modelu založeném na anglickém jazyce nabízí autor projektu CMU Sphinx webovou službu, která na základě seznamu příkazů vytvoří příslušný jazykový model a slovník. [69]

Akustický model Knihovna CMU Sphinx disponuje několika akustickými modely v několika jazycích pro různé způsoby záznamu rozpoznávaného zvuku. V případě, že je základní model nevyhovující, ať už z důvodu zkreslení zvuku při nahrávání, nebo vlivem přízvuku uživatele, je možné provést adaptaci modelu. K té je potřeba použití nahraných zvukových vzorků daného uživatele pořízených na zařízení, na kterém bude probíhat záznam při rozpoznávání [70]. Aplikace Zoomera umožňuje použití i upraveného akustického modelu (vize Sekce 6.4.4). S několika takovými modely bylo provedeno testování s uživateli (vize Sekce 8.2). V případě potřeby specifického akustického modelu je možné trénovat zcela nový model za použití nástrojů projektu CMU Sphinx a zvukových ukázek daného jazyka.

6.4.3.3. Implementace pro systém Android

Následující část popisuje základní konfiguraci a implementaci modulu pro rozpoznávání řeči v aplikaci Zoomera. Kostra návrhu vychází z referenční implementace pro Android od autorů CMU Sphinx [67, 68].

Inicializace dekodéru Pomocí instance objektu třídy Config se vytvoří konfigurace knihovny Pocketsphinx. Objekty třídy Config se předá v konstruktoru při vytváření objektu Decoder, který zajišťuje samotné rozpoznávání (viz Obrázek 50).

Parametry *-hmm*, *-dict* a *-lm* nastavují cesty k souborům s definicemi rozpoznávaného jazyka. Parametr *-rawlogdir* nastavuje složku pro zápis logovacích souborů, *-samprate* udává vzorkovací frekvenci vstupního audia v Hz. Hodnoty *-maxhmmpf*, *-ds* a *-bestpath* detailně nastavují způsob rozpoznávání a byly převzaty z [67, 68] nebo nastaveny podle [71].

```
1  // String path je cesta ke slozce se soubory jazykoveho modelu
2  Config config = new Config();
3  config.setString("-hmm",       path + "hmm/en_US/hub4wsj_sc_8k");
4  config.setString("-dict",      path + "lm/zoomera/dictionary.dic");
5  config.setString("-lm",        path + "lm/zoomera/language-model.dmp");
6  config.setString("-rawlogdir", path);
7  config.setFloat("-samprate", 16000.0);
8  config.setInt("-maxhmmpf", 10000);
9  config.setInt("-ds", 1);
10 config.setBoolean("-bestpath", true);
11
12 Decoder decoder = new Decoder(config);
```

Obrázek 50. Zdrojový kód, který konfiguruje a vytvoří dekodér pro rozpoznávání řeči.

Použití dekodéru Použití dekodéru ilustruje Obrázek 51. Dekodér se spustí zavoláním metody *Decoder.startUtt()*. Získáme zvukový fragment v short poli *buffer* z fronty objektu třídy AudioRecorder (viz 6.4.1.1). Fragment *buffer* předáme v parametru metody Decoder.processRaw(), jejímž výsledkem je hypotéza *Hypothesis*, ze které můžeme získat rozpoznaná slova metodou *Hypothesis.getHypStr()*. Pro kontinuální detekci je kód na Obrázku 51 prováděn v cyklu.

```
1  this.decoder.startUtt();
2  short[] buffer = this.audioRecorder.queueTake();
3  this.decoder.processRaw(buffer, buffer.length, false, false);
4  Hypothesis hypothesis = this.decoder.getHyp();
5  String hypStr = hypothesis.getHypstr();
```

Obrázek 51. Zdrojový kód, demonstruje použití dekodéru pro rozpoznávání řeči.

Kontinuální detekce Výše uvedený způsob získání hypotézy funguje inkrementálně. Znamená to, že si dekodér postupně sestavuje pravděpodobnostní model pro rozpoznání slov v poskytnutých audio fragmentech tak, že po každém zavolání metody *Decoder.processRaw()* získáme hypotézu pro všechny dosud zpracované audio fragmenty. Tento způsob je výhodný pro rozpoznávání časově omezeného audio úseku. Pokud ale chceme provádět rozpoznávání nepřetržitě, uvažujeme teoreticky nekonečný audio úsek, je nutné implementaci provést tak, že se dekodér pravidelně restartuje a staré výsledky rozpoznávání se tak nekumulují a nezaplňuje se zbytečně paměť zařízení.

Dekodér je restartován podle tří kritérií:

- Prvním je délka již rozpoznaného řetězce. Pokud tato délka překročí počet 20 slov a zároveň je stavový automat pro validaci příkazů v počátečním stavu (viz 6.4.2.2 a 6.4.3.4), je dekodér restartován. Podmínka pro stavový automat je zastoupena, protože pokud je automat v počátečním stavu, znamená to, že řetězec neobsahuje žádný nedokončený příkaz. Uvedená hodnota byla vybrána, protože se předpokládá, že délka příkazů bude mnohem kratší, a tedy vše, co překročí stanovenou mez, je považováno za chybný vstup.
- Druhým kritériem je doba, po kterou je rozpoznávací modul neaktivní z důvodu chybějícího uživatelského hlasového vstupu. Doba se měří jako počet po sobě v řadě provedených zpracování audio fragmentů rozpoznávacím modulem, jejichž výsledek detekce je prázdný. Tato hodnota je v aplikaci nastavena na 100 iterací, po kterých následuje restart dekodéru. Byla stanovena experimentálně pozorováním stavu paměti zařízení. Doba, po kterou může být dekodér aktivní, je mnohem delší, ale pro účely použití jednoduchých příkazů v aplikaci Zoomera je postačující.

- Třetím kritériem je samotné rozpoznávání příkazů. Ve chvíli, kdy je rozpoznán příkaz, je dekodér automaticky restartován.

Běh v samostatném vlákně a jeho řízení Protože modul pro rozpoznávání řeči je výpočetně náročný, je nutné, aby byl spuštěn v samostatném vlákně. Princip fungování vlákna je inspirován referenční implementací použití knihovny CMU Sphinx v aplikaci pro systém Android od autora zmíněné knihovny [67].

```
1   private enum State { IDLE, LISTENING };
2   private enum Event { NONE, START, STOP, RELEASE };
3   private State state;
4   private Event message;
5
6   private boolean running;
7
8   // ...
9
10  @Override
11  public void run() {
12     this.running = true;
13     this.state = State.IDLE;
14     while (running) {
15        if (reinit) {
16           // Provedeni pripadne reinicializace dekoderu
17        }
18
19        // Synchronizace pomoci posilani zprav
20        Event event = Event.NONE;
21        synchronized (message) {
22           event = this.message;
23           if (this.state == State.IDLE && event == Event.NONE) {
24              try {
25                 this.message.wait();
26                 event = this.message;
27              } catch (InterruptedException e) {
28                 event = Event.RELEASE;
29              }
30           }
31           this.message = Event.NONE;
32        }
33
34        // Provedeni dane akce rizeni
35        switch (event) {
36           case NONE: { break; }
37           case START: { break; }
38           case STOP: { break; }
39           case RELEASE: { break; }
40        }
41
42        // Zde se provadi volani modulu pro rozpoznavaci reci
43        if (this.state == State.LISTENING) {
44
45        }
46     }
47  }
```

Obrázek 52. Zdrojový kód, který definuje funkci vlákna obsluhujícího rozpoznávání řeči.

Ukázka zdrojového kódu na Obrázku 52 demonstruje základní fungování vlákna pro ovládání rozpoznávání řeči. Vlákno je řízeno pomocí proměnné *State state*, která udává stav rozpoznávání (*IDLE* - v klidu, *LISTENING* - aktivní rozpoznávání), a proměnné *Event message*, pomocí které je možné tomuto vláknu poslat zprávu (*NONE* - žádná akce, *START* - spuštění rozpoznávání, *STOP* - zastavení rozpoznávání, *RELEASE* -

ukončení vlákna), která definuje jeho další chování.

6.4.3.4. Validace rozpoznaných slov

Modul rozpoznávání řeči na základě vstupu rozpozná řetězec obsahující slova, která vzhledem k použitému jazykovému a akustickému modelu jsou nejpravděpodobnější interpretací zaznamenané řeči. Výstup se skládá pouze z slov obsažených ve slovníku jazykového modelu. V případě aplikace Zoomera platí, že slovník použitého jazykového modelu, který je tvořen pouze slovy obsaženými v příkazech (viz Sekce 6.4.2.3), je o mnoho menší než slovník anglického jazyka. V takovém případě jsou slova neobsažená v jazykovém modelu aplikace interpretována rozpoznávacím modulem jako slova, která jsou v modelu obsažena a jsou jim z hlediska použitého jazykového a akustického modelu nejblíže. Výsledkem rozpoznávání tak bývá řetězec, který se skládá ze slov, a v jeho podřetězci se může nacházet hledaný příkaz.

Stavový automat Pro nalezení příkazů, které jsou obsaženy v řetězci se používá stavový automat, který se rozhoduje na základě vstupu, který obsahuje kontrolovaný řetězec. Použitý automat je detailně popsaný v Sekci 6.4.2.2. Automat je volán rozpoznávacím modulem při každém novém výsledku rozpoznávání. Pokud se automat na základě vstupu dostane do koncového stavu, provede se odpovídající příkaz.

Validace příkazů Protože se provedení příkazů může spustit na základě jakéhokoli nového výsledku rozpoznávání, je nutné příkazy validovat. Důvodem je fakt, že může dojít k rozpoznání špatného příkazu ve chvíli, kdy ještě nejsou zpracovány všechny fragmenty audio záznamu obsahující znění daného příkazu.

Validace se provádí tak, že pokud je na základě rozpoznání řeči stanoven příkaz, který se má spustit, čeká se 400 ms, zda nebude na výstupu z rozpoznávacího modulu jiný příkaz. To zamezí tomu, aby nedokončená řeč byla nesprávně rozpoznána. Délka čekání byla určena experimentálně tak, aby správnost rozpoznávání byla vysoká a zároveň aby odezva hlasového ovládání umožňovala pohodlnou práci s aplikací.

6.4.4. Úprava akustického modelu

Knihovna CMU Sphinx umožňuje úpravu dostupného akustického modelu a poskytuje na tento proces také své nástroje. Úprava akustického modelu může zlepšit rozpoznání řeči na základě poskytnutých ukázkových vzorků hlasu a jejich ohodnocení vzhledem k použitému jazykovému modelu. Ke každému vzorku je přiřazen textový řetězec, který definuje obsah zvukového vzorku. Adaptace upraví model tak, aby přesněji odpovídal konkrétnímu uživateli, který zvukové ukázky nahrál, ale také aby odpovídal konfiguraci a možnostem nahrávacího zařízení, které bylo použito pro záznam vzorků. [72]

V aplikaci Zoomera je použit výchozí akustický model pro US angličtinu, který je dodáván autorem projektu CMU Sphinx. Během testování (viz Sekce 8.2) byly také použity upravené akustické modely pro jednotlivé uživatele.

6.4.4.1. Použití nástrojů CMU Sphinx

Pro adaptaci akustického modelu je v projektu CMU Sphinx několik nástrojů v projektu SphinxBase a SphinxTrain. Na základě [72] byl vytvořen skript pro systém GNU Linux, který umožňuje za použití uvedených nástrojů provést adaptaci modelu (viz Přílohy - Obrázek 78).

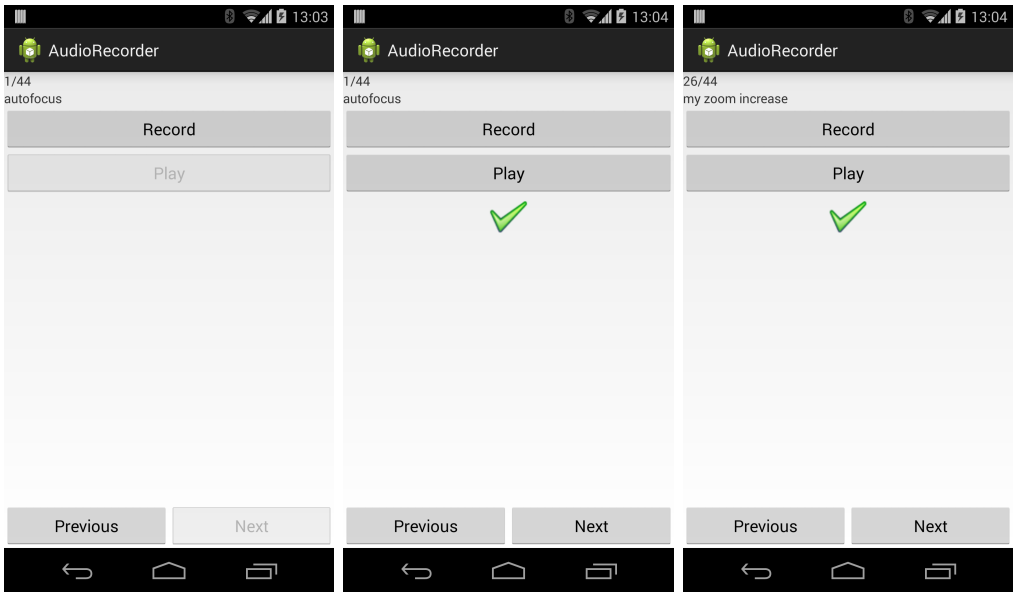
Pro spuštění skriptu je nutné, aby se složka *hub4wsj_sc_8k* nacházela ve stejném umístění s jazykovým modelem určeným k úpravě, soubor *zoomera.listoffiles* se seznamem názvů souborů zvukových ukázek, *zoomera.transcription* s přiřazenými textovými řetězci a názvy odpovídajících souborů ve formátu: <s> *TEXTOVÝ ŘETĚZEC* </s> (*název_souboru*). Ve stejné složce se také musí nacházet všechny uvedené zvukové ukázky ve formátu WAVE. Po spuštění skriptu bude ve složce *hub4wsj_sc_8kadapt* vytvořen upravený akustický model.

6.4.4.2. Použití upraveného modelu

V zařízení ve složce */sdcard/edu.cmu.pocketsphinx/hmm/en_US/hub4wsj_sc_8k* se nachází akustický model použitý v aplikaci Zoomera. Pro použití upraveného modelu je nutné obsah této složky vymazat a nahradit jí obsahem složky *hub4wsj_sc_8kadapt*, která vznikla úpravou akustického modelu. Po restartování aplikace Zoomera bude v modulu rozpoznávání řeči aktivní upravený akustický model.

6.4.5. Aplikace pro záznam vzorků hlasu

Pro záznam zvukových ukázek hlasu, které mohou být použity pro úpravu akustického modelu, byla vytvořena aplikace *AudioRecorder*, která využívá třídy *AudioRecord* (viz Sekce 6.4.1.1) pro záznam zvuku. Zvukové ukázky jsou ukládány do paměti zařízení ve formátu WAVE (viz Sekce 6.4.5.1) s vzorkovací frekvencí 16000 Hz, jedním zvukovým kanálem a 16 bity na vzorek.



Obrázek 53. Grafické uživatelské rozhraní aplikace *AudioRecorder* pro nahrávání vzorků hlasu. Vlevo - výchozí stav GUI před nahráním vzorku hlasu pro jedno slovo ze slovníku. Uprostřed - stav GUI po nahrání vzorku hlasu. Vpravo - stav GUI po nahráním vzorku hlasu pro jeden příkaz.

Do aplikace je na vstupu vložen seznam slov a frází, které se mají nahrát. Aplikace na základě tohoto seznamu vytvoří GUI, kterým je uživatel provázen k postupnému nahrávání ukázek. Číslo aktuální ukázky, počet všech aktuálních ukázek a text, který má uživatel nahrát je uveden v levém horním rohu GUI. Každou ukázku je možné nahrát (tlačítko *Record*) a nahranou ukázku je možné přehrát (tlačítko *Play*). Mezi

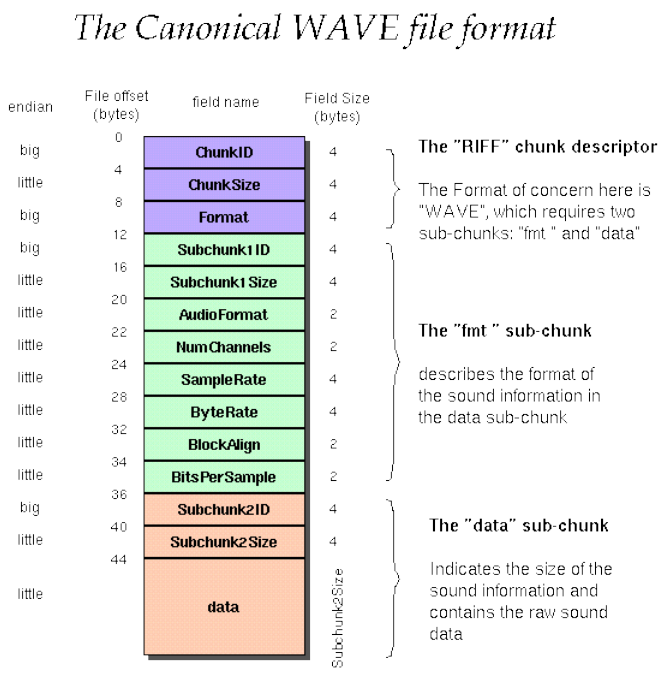
jednotlivými obrazovkami pro nahrání ukázek se přepíná pomocí tlačítek *Previous* a *Next*.

Výsledkem jsou soubory ve formátu WAVE uložené v paměti zařízení ve složce /sdcard/Zoomera/AudioRecorderFiles/*datum*/ ve tvaru *název-slova-nebo-příkazu*.wav.

6.4.5.1. WAVE formát

V systému Android není možné nahrávat zvuk přímo do formátu WAVE. První možností je použití třídy *MediaRecorder*, ten však umožňuje nahrávání pouze do ztrátových formátu, což je nevhodné pro použití v systému rozpoznávání řeči. Druhou možností je použití třídy *AudioRecord* (viz Sekce 6.4.1.1) stejně, jako je použito v aplikaci Zoomera. Tato metoda umožňuje nahrávání zvuku do datového pole v operační paměti zařízení. Pole může být typu byte, short nebo ByteBuffer.

Výstup z nahrávání instancí třídy *AudioRecord* jsou zvuková data převedená pulzně kódovou modulací (PCM). Tato data nenesou žádné informace o vlastnostech zvuku (vzorkovací frekvence, počet zvukových kanálů a počet bitů na vzorek), který obsahují. Pokud na jejich začátek přidáme hlavičku definovanou podle standardu pro formát WAVE (viz Obrázek 54), získáme tak WAVE soubor, který lze přehrát v softwarových přehrávačích a umožňuje také použití při úpravě akustického modelu rozpoznávání řeči.



Obrázek 54. Struktura hlavičky WAVE formátu souboru. [73]

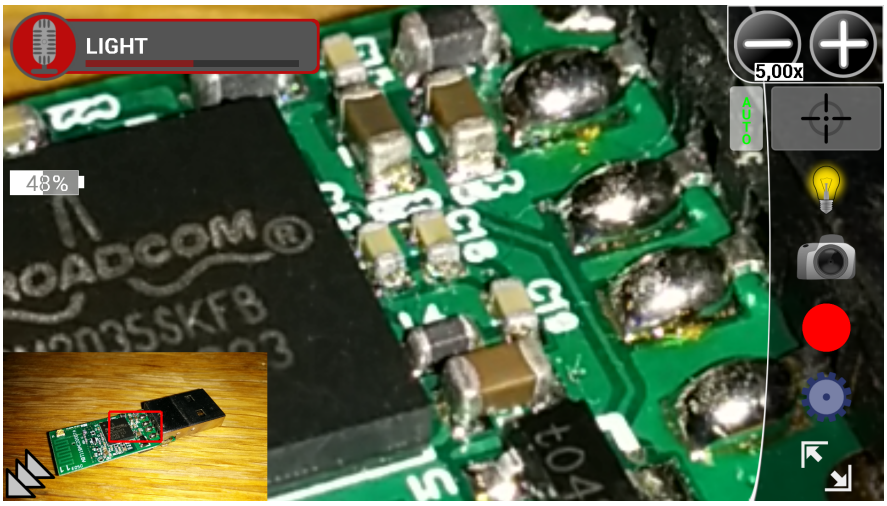
Pro převod zvukových dat z formátu PCM do formátu WAVE byla vytvořena sada funkcí, které to umožňují. Na Obrázku 54 je možné v levém sloupci vidět, s jakou endianitou jsou jednotlivé části hlavičky souboru WAVE uloženy. Pro tento účel byla vytvořena třída *FileOperations*, která umožňuje ukládat do souboru datové typy a datová pole s nastavením příslušné endianity. Příslušné funkce jsou použity v metodě *writeWAVHeader()* (viz Přílohy - Obrázek 79). Tato metoda umožňuje zápis hlavičky WAVE souboru podle vstupních parametrů tak, jak je definována v [73].

7. Grafické uživatelské rozhraní

Tato kapitola obsahuje informace ohledně důležitých prvků grafického uživatelského rozhraní aplikace Zoomera. Přidání nových funkcí si vyžádalo i náležitou úpravu uživatelského rozhraní aplikace. Bylo přidáno ovládání modulu rozpoznávání řeči, ovládání nahrávání videa, nastavení nahrávání videa, nastavení funkcí aplikace, ovládání Bluetooth audio tunelu. Bylo přepracováno ovládání digitálního zoom, byl přidán indikátor stavu baterie a nastavení jasu obrazovky.

7.1. Základní obrazovka aplikace Zoomera

Aplikace Zoomera má jedinou obrazovku, která obsahuje všechny potřebné ovládací prvky (viz Obrázek 55). Po celé ploše obrazovky je vykreslen živý náhled z kamery zařízení. Po pravé straně jsou umístěny nejdůležitější ovládací prvky, jako je ovládání zoom, ostření, ovládání světla, fotografování, nahrávání videa, nastavení aplikace a tlačítko pro skrytí všech ovládacích prvků.



Obrázek 55. Grafické uživatelské rozhraní se zapnutým hlasovým ovládáním. Obrázek ukazuje situaci při hlasovém zadání příkazu pro rozsvícení LED.

V pravém horním rohu je ovládací prvek modulu hlasového ovládání a indikátor stavu baterie. Mezi těmito dvěma prvky se během nahrávání zobrazuje indikátor (viz 7.5). V levém dolním rohu je tlačítko pro zobrazení/skrytí miniatury obrazu. Ve stejných místech se zobrazí i samotná miniatura.

7.2. Zobrazení na celou obrazovku

S uvedením Androidu verze 4.4 (API 19, KitKat) se objevila možnost zobrazení aplikace na celou obrazovku bez navigačního panelu [74]. Ovládací prvky se zobrazí samy,

když jsou potřeba, nebo je může zobrazit uživatel sám tažením kolmo na okraj displeje ve směru doprostřed displeje v místech, kde se panel běžně nachází. Implementaci popisuje Obrázek 56.

```
1   @Override
2   public void onWindowFocusChanged(boolean hasFocus) {
3      super.onWindowFocusChanged(hasFocus);
4      if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
5          if (hasFocus) {
6            this.getWindow().getDecorView().setSystemUiVisibility(
7                      View.SYSTEM_UI_FLAG_LAYOUT_STABLE
8                      | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
9                      | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
10                     | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
11                     | View.SYSTEM_UI_FLAG_FULLSCREEN
12                     | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY);
13         }
14     }
15  }
```

Obrázek 56. Ukázka zdrojového kódu, který umožní zobrazit aplikaci na celou obrazovku bez navigačního panelu. Metoda je umístěná v potomku třídy *Activity*.

7.3. Ovládání zoom

Ovládání zoom je prováděno pomocí tlačítek se symbolem plus a minus (viz Obrázek 57). Tlačítko pro zmenšení digitálního zoom (znak minus), je z části překryto textovým polem, které zobrazuje aktuální hodnotu zoom.

Obrázek 57. Ovládací prvky digitálního zoom. Horní obrázek zobrazuje výchozí stav, dolní stav, kdy je zobrazen i posuvník.

Detailnější nastavení zoom je dostpné ve chvíli, kdy je po stisku jednoho z tlačítek zoom zobrazen posuvník. Ten slouží k rychlejšímu nastavení zoom díky grafickému zobrazení celé škály. Posuvník je zobrazen pouze po dobu čtyř sekund, poté je znovu skryt, dokud není znovu stisknuto jedno z tlačítek ovládání zoom. Pro přiblížení obrazu je možné také použít hardwarová tlačítka pro ovládání hlasitosti, nebo tažení dvou prstů na obrazovce směrem od sebe a k sobě.

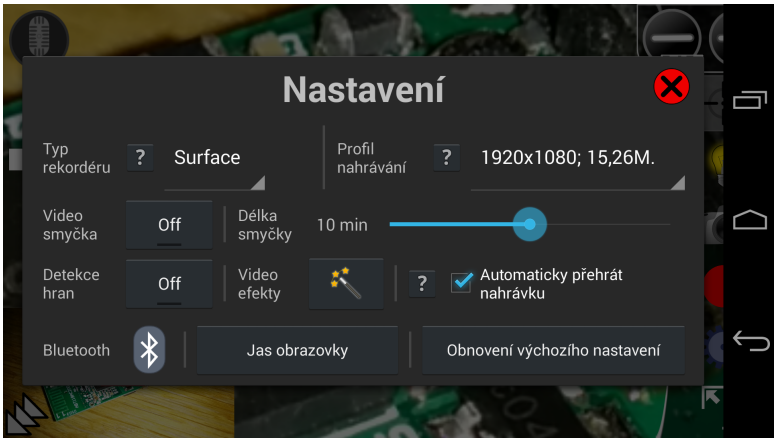
Spolu s nastavením přiblížení obrazu je také nastavena plocha obrazu, kterou má systém uvažovat při ostření kamery a nastavování expozice (viz Obrázek 58). Takové nastavení umožňuje brát v ohled fakt, že uživatel při nastavení digitálního zoom nevidí celý obraz, ale jen jeho výřez. Ostření a měření expozice je nastaveno pouze na tento výřez, a tak může nastavení kamery reagovat jen na to, co uživatel vidí.

```
Camera camera;
// Definice plochy a jeji pridani do listu
Camera.Area area = new Camera.Area(new Rect(left, top, right, bottom),
    weight);
List<Camera.Area> listAreas = new LinkedList<Camera.Area>();
listAreas.add(area);
Parameters parameters = camera.getParameters();
// Kontrola, zda jsou plochy pro ostreni podporovany
if (parameters.getMaxNumFocusAreas() > 0) {
   parameters.setFocusAreas(listFocusAreas);
}
// Kontrola, zda jsou plochy pro mereni expozice podporovany
if (parameters.getMaxNumMeteringAreas() > 0) {
   parameters.setMeteringAreas(listMeteringAreas);
}
camera.setParameters(parameters);
```

Obrázek 58. Ukázka zdrojového kódu, který umožní nastavit plochy, které mají být uvažované při ostření a nastavování expozice.

7.4. Nastavení aplikace

S rostoucím počtem možných nastavení bylo nutné vytvořit samostatnou nabídku, která bude všechny volby sdružovat (viz Obrázek 59). Nabídka nastavení se vyvolá tlačítkem v podobě ozobeného kola v pravém sloupci ovládacích prvků hlavní obrazovky (viz Obrázek 55). Nabídka obsahuje nastavení nahrávání videa, nastavení nekonečné smyčky, Bluetooth, nastavení efektů obrazu a možnost pro obnovení výchozího nastavení. Všechny hodnoty se ukládají do paměti zařízení, a tak jsou dostupné i po tom, co byla aplikace vypnuta.



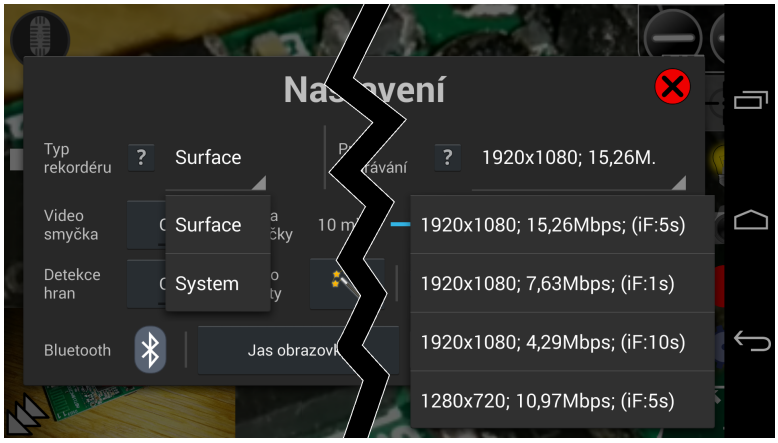
Obrázek 59. Grafické uživatelské rozhraní se zobrazeným nastavením.

7.4.1. Nastavení nahrávání videa

Nabídka nastavení umožňuje zvolit požadovanou konfiguraci pro nahrávání videa z kamery. Existují dva parametry, které je možné nastavit. Prvním je nastavení typu rekordéru, na výběr je mezi systémovým nahráváním (System) a nahráváním pomocí OpenGL ES textury a třídy *MediaCodec* (viz Obrázek 60 - vlevo). Druhým parametrem je nastavení profilu nahrávání (viz Obrázek 60 - vpravo). Ten umožňuje zvolit předdefinovanou konfiguraci nahrávání, jako je rozlišení nebo datový tok. Obsah seznamu profilů je závislý na zvoleném typu rekordéru.

Dále je též možné nastavit, zda se má po skončení nahrávání spustit automatické

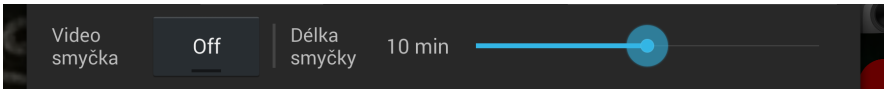
přehrávání záznamu (viz Obrázek 67). To se provede zatržením ovládacího prvku s nápisem *"Automaticky přehrát nahrávku"* v nabídce nastavení.



Obrázek 60. Možnosti nastavení typu rekordéru a profilu nahrávání.

7.4.2. Nastavení nekonečné smyčky

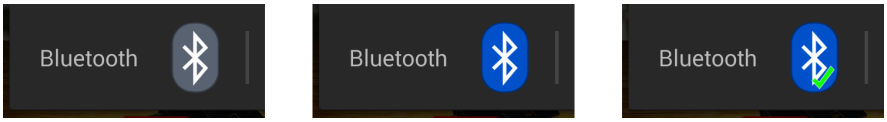
Nahrávání v nekonečné smyčce je možné zapnout tlačítkem uvedeným na Obrázku 61 a 59. Nastavení má jeden parametr, což je požadovaná délka smyčky. Ta se nastavuje posuvníkem a jsou k dispozici předdefinované hodnoty od 10 sekund do 240 minut. Nastavení je viditelné pouze, když je vybrán rekordér typu Surface. Jiný rekordér nahrávání ve smyčce nepodporuje.



Obrázek 61. Možnosti nastavení nahrávání ve smyčce.

7.4.3. Bluetooth audio tunel

Pod ovládáním nastavení nekonečné smyčky se nachází tlačítko pro spouštění Bluetooth audio tunelu. V případě, že je Bluetooth spojení zakázáno, je ovládací prvek zbarvený do šeda (viz Obrázek 62 - levý). Dotykem přejde do stavu spojení povoleno a dojde k spojování (viz Obrázek 62 - prostřední). V případě, že dojde k úspěšnému spojení, přejde ovládací prvek do třetího stavu (viz Obrázek 62 - pravý).

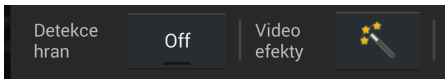


Obrázek 62. Ovládací prvek bluetooth spojení. Levý - spojení zakázáno, Prostřední - spojení povoleno, Pravý - spojení navázáno.

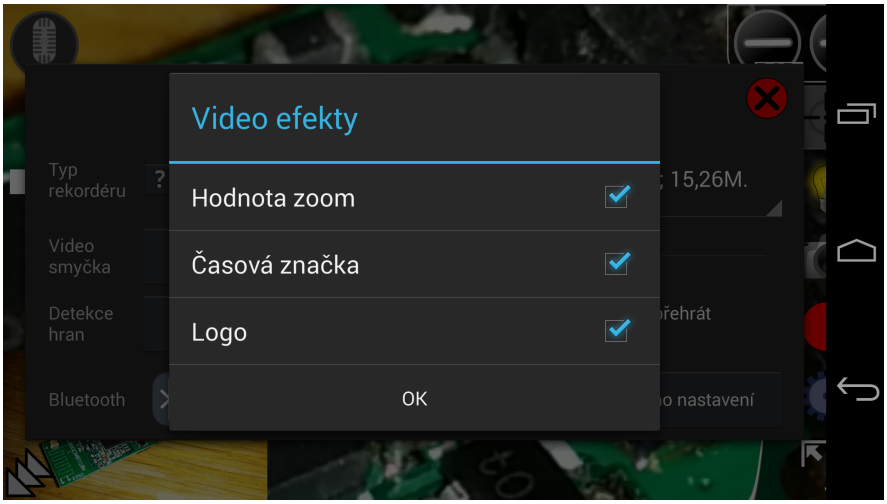
7.4.4. Nastavení efektů obrazu

Napravo od ovládacího prvku Bluetooth se nachází dvojice tlačítek, které umožňují nastavovat efekty obrazu/videa (viz Obrázek 63). Levé tlačítko zapíná/vypíná detekci hran. Ta se zobrazí na obrazovce a se nahraje do případného videa. Pravým tlačítkem se

otevře nabídka (viz Obrázek 64), které umožňuje nastavení efektů videa (viz Kapitola 5.6.1.2). Tyto efekty se projeví pouze v zaznamenaném videu.

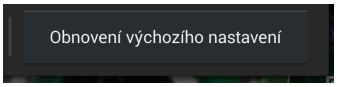


Obrázek 63. Ovládací prvky pro nastavení obrazových efektů. Levý - detekce hran, Pravý - efekty videa.



Obrázek 64. Nabídka pro zvolení efektů videa.

7.4.5. Obnovení výchozího nastavení



Obrázek 65. Tlačítko pro obnovení výchozího nastavení.

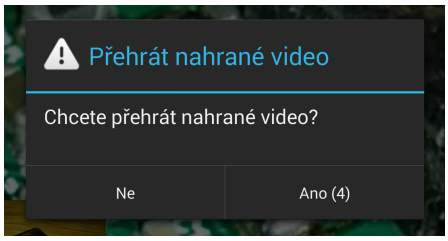
V pravém dolním rohu nabídky nastavení se nachází tlačítko (viz Obrázek 65), které umožňuje nastavit všechny uložené hodnoty do výchozího stavu. Před vymazáním hodnot je uživatel dotázán, zde chce akci opravdu provést. Obnovení výchozího nastavení není možné vrátit zpět.

7.5. Indikátor nahrávání

Pro zobrazení informace o právě probíhajícím nahrávání videa slouží indikátor na Obrázku 66. Ten vykresluje čas, který uběhl od začátku nahrávání spolu s blikajícím červeným bodem. Ovládací prvek nahrávání reaguje na nastavení nahrávání ve smyčce. Pokud je nahrávání ve smyčce vypnuté, je zobrazen červený bod, pokud je zapnuté, zobrazuje se červený bod se znakem položeného čísla osm (viz Obrázek 66).

Po ukončení nahrávání se zobrazí dialog s dotazem, zda má být právě nahrané video přehráno (viz Obrázek 67). Pokud uživatel nezareaguje do pěti sekund, přehrávání se spustí automaticky.

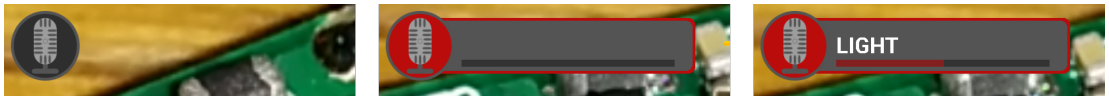
Obrázek 66. Vlevo je zobrazen indikátor nahrávání videa. Vpravo se nacházejí dvě varianty ovládacího prvku nahrávání. Horní se zobrazí v případě, že je nastaveno běžné nahrávání, dolní, když je aktivováno nahrávání ve smyčce.



Obrázek 67. Dialog, který se dotazuje na přehrání videa po ukončení nahrávání.

7.6. Hlasové ovládání

Hlasové ovládání aplikace Zoomera se spouští pomocí tlačítka s ikonou mikrofonu v levém horním rohu GUI (viz Obrázek 55 a 68). Ve chvíli, kdy je hlasové ovládání neaktivní, je zobrazena pouze ikona mikrofonu (viz Obrázek 68 - levý). Dotykem na ikonu dojde ke spuštění hlasového ovládání, objeví se šedivý obdélník s červeným orámováním, který zobrazuje právě zadané příkazy a úroveň hlasitosti zaznamenávaného audia (viz Obrázek 68 - prostřední). Zadaný příkaz se zobrazí uvnitř ovládacího prvku hlasového ovládání (viz Obrázek 68 - pravý).



Obrázek 68. Nově přidaná skupina ovládacích prvků a indikátorů. Obrázky zobrazují možné stavy ovládacího prvku hlasového ovládání (ikona mikrofonu). Levý obrázek zobrazuje tento ovládací prvek v případě, že je hlasové ovládání neaktivní. Prostřední obrázek zobrazuje stav při spuštěném hlasovém ovládání. Pravý obrázek zobrazuje případ, kdy byl rozpoznán příkaz na rozsvícení LED.

7.7. Indikátor stavu baterie

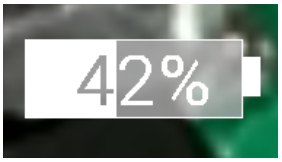
Aplikace je nastavena tak, aby se při jejím běhu nezobrazoval systémový stavový panel. Protože je aplikace Zoomera náročná na výkon zařízení, vybíjí se baterie rychleji. Z toho důvodu byl přidán indikátor stavu baterie (viz Obrázek 70), který je umístěn pod ovládacím prvkem hlasového ovládání (viz Obrázek 55). Prvek využívá systémové třídy *IntentFilter* k získání objektu, který poskytuje aktuální stav baterie [75] (viz ukázka zdrojového kódu na Obrázku 69).

```
1  IntentFilter intentFilter = new
       IntentFilter(Intent.ACTION_BATTERY_CHANGED);
2  this.batteryStatusIntent = context.registerReceiver(null, intentFilter);
3  int level =
       this.batteryStatusIntent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
4  int scale =
       this.batteryStatusIntent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
```

Obrázek 69. Ukázka zdrojového kódu, který vytvoří objekt poskytující aktuální stav baterie.

7.7.1. Zobrazení hladiny nabití

Indikátor baterie zobrazuje hladinu nabití pomocí vyplnění ikony baterie obdélníkem o odpovídající šířce. Přes ikonu baterie se dále vykresluje text, který udává hladinu baterie v procentech. Aby byl text dobře čitelný i v případě, že přes něj zasahuje obdélníkové vyplnění jen částečně, zobrazuje se část textu v barvě pozadí ikony a část textu v barvě výplně tak, aby byl text kontrastní. Toho je dosaženo aplikací správného Porter Duff pravidla, které definuje způsob kompozice dvou obrazových ploch [15].



Obrázek 70. Indikátor hladiny nabití baterie zařízení.

7.7.2. Nastavení jasu

V případě, že se uživatel dotkne plochy na displeji v místech, kde se nachází indikátor stavu baterie, dojde k zobrazení dialogového okna s možností upravit jas displeje. Zároveň se v nabídce nastavení (viz Obrázek 59) nachází tlačítko, které má stejnou funkčnost. Jasové hodnoty je možné nastavit ve 100 krocích od hodnoty 0 do hodnoty 100, kde obě meze jsou nastaveny výrobcem zařízení a jsou také závisle na použitých technologiích v displeji. Nastavení je platné pouze pro aktuálně spuštěnou aplikaci a nemá vliv na nastavení jasu displeje v systému a ostatních aplikacích.

```
1  // Nastaveni hodnoty jasu; brightness >= 0.0f && brightness <= 1.0f
2  WindowManager.LayoutParams lp = activity.getWindow().getAttributes();
3  lp.screenBrightness = brightness;
4  activity.getWindow().setAttributes(lp);
5  // ...
6  // Ziskani vychozi systemove hodnoty jasu.
7  int defaultSystemBrightness = Settings.System.getInt(
8                                          activity.getContentResolver(),
9                                          Settings.System.SCREEN_BRIGHTNESS
10                                 )/255.0f;
```

Obrázek 71. Ukázka zdrojového kódu, který umožní nastavit jas displeje.

Obrázek 71 zobrazuje zdrojový kód pro nastavení jasu displeje. Využívá se pro to systémová třída *LayoutParams* a její proměnná *screenBrightness* [76]. Ta může nabývat hodnot od 0 do 1, kde 0 představuje nejnižší možný jas a 1 nejvyšší možný jas. V případě, že je hodnota menší než 0, použije se výchozí systémové nastavení, které se získá metodou *getInt* ve třídě *Settings.System* [77].

8. Testování

Tato kapitola obsahuje veškeré testovací procedury, které byly na aplikaci Zoomera provedeny. Byly testovány metody a dílčí principy týkající se nahrávání videa. Modul pro ovládání aplikace hlasovými povely byl testován s uživateli, důraz byl kladen na ověření funkčnosti rozpoznávacího procesu. U hlasového rozpoznávacího modulu byly dále zkoumány výpočetní a paměťové nároky.

8.1. Nahrávání videa

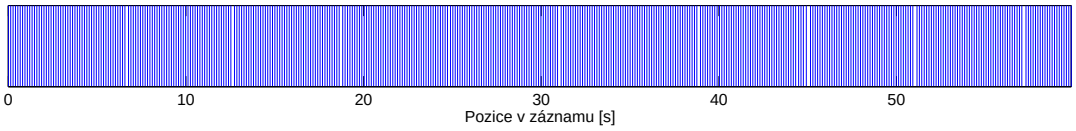
Níže jsou zastoupeny výsledky všech testování týkající se nahrávání videa na platformě Android. Testování byla prováděna na zařízení LG Nexus 5 popsaném v 4.3.1. Týkají se systémového nahrávání, funkce *glReadPixels()*, nahrávání pomocí knihovny FFmpeg a kopie snímků a nahrávání pomocí OpenGL textury a třídy *MediaCodec*. Výsledky těchto testů demonstrují důvody, proč byly některé metody do aplikace Zoomera implementovány a některé ne.

8.1.1. Systémové nahrávání

Nahrávání videa z kamery pomocí systémové třídy *MediaRecorder* je popsáno v Kapitole 5.4.2.1. Testování bylo prováděno nahráním testovacího záznamu o délce jedné minuty touto metodou v aplikaci Zoomera na zařízení LG Nexus 5. Následně byl záznam analyzován programem Avidemux[78], jehož pomocí byla zjištěna přesná délka záznamu a počet video snímků, které obsahuje. Díky nim mohla být vypočítána průměrná snímková frekvence videa udávaná ve snímcích za vteřinu (Frames per second - fps). Tato hodnota mimo jiné udává jak plynule jsou případné pohyby na videu zaznamenány. Čím vyšší snímková frekvence, tím je pro diváka video věrnější.

Video profil	**Počet snímků**	**Délka záznamu [m:s]**	**Snímková frekvence [fps]**
320×240, 15 fps, 0.12 Mbps	1785	0:59.798	**29.85**
720×480, 30 fps, 5.72 Mbps	1791	1:00.000	**29.85**
1280×720, 30 fps, 11.44 Mbps	1771	0:59.966	**29.53**
1920×1080, 30 fps, 16.21 Mbps	1765	0:59.865	**29.48**

Tabulka 5. Výsledky testování systémového nahrávání.



Obrázek 72. Znázornění časových značek snímků videa zaznamenaném systémovým nahráváním v rozlišení 1920×1080. Je zobrazen každý čtvrtý snímek.

Byly testovány čtyři video profily definované rozlišením, výchozí snímkovou frekvencí a datovým tokem (viz Tabulka 5). Všechny čtyři testované profily nejevily žádné zásadní snížení snímkové frekvence a všechny se pohybovaly v rozmezí od 29.48 do 29.85 fps. Od tohoto způsobu nahrávání se takový výstup očekával, protože metoda je součástí systému a může tak být optimalizována jak výrobcem zařízení, tak vývojářem systému Android. Na Obrázku 72 jsou graficky znázorněny časové značky snímků videa nahraného touto metodou. Časové značky netvoří shluky, jsou rovnoměrně rozložené po celé délce záznamu. Jedná se o stálou snímkovou frekvenci s občasnými výpadky snímků.

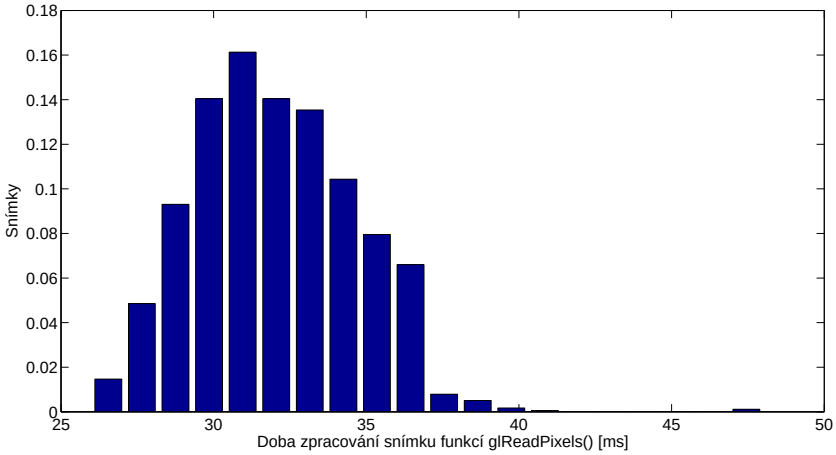
8.1.2. Funkce glReadPixels

Funkce *glReadPixels()* určená pro čtení obsahu povrchu vykreslovaného pomocí OpenGL je popsána v Kapitole 5.3.3. Funkce byla uvažována jako metoda pro získání kopie snímků z kamery vekreslených pomocí OpenGL v Kapitole 5.4.2.3. Bylo prováděno měření doby potřebné pro přečtení jednoho snímku o daném rozlišení popsané na Obrázku 73.

```
1  // Alokace pameti pro cteni snimku o rozliseni readAreaWidth x
       readAreaHeight. Nasobek 3 je z dovodu pouziti RGB barevneho prostoru s
       1 bytem na barevny kanal.
2  ByteBuffer readBuffer =
       ByteBuffer.allocateDirect(3*readAreaWidth*readAreaHeight);
3  // ...
4  long readStartTime = System.currentTimeMillis();
5  // Precteni snimku z obrazovky.
6  GLES20.glReadPixels(0, 0, readAreaWidth, readAreaHeight, GLES20.GL_RGB,
       GLES20.GL_UNSIGNED_BYTE, readBuffer);
7  long readTime = System.currentTimeMillis()-this.readStartTime;
```

Obrázek 73. Ukázka způsobu měření doby potřebné pro přečtení bloku obrazu o daném rozlišení finkcí glReadPixels().

Byly testovány čtyři rozlišení čtené plochy (viz Tabulka 6). Každý test trval jednu minutu, během které byla pravidelně volána funkce *glReadPixels()*, byla měřena doba potřebná pro jedno čtení. Jako výsledná hodnota testu byla uvedena průměrná hodnota jednoho čtení.



Obrázek 74. Histogram naměřených časů potřebných pro přečtení snímku videa v rozlišení 320×180 pomocí funkce *glReadPixels()*.

Na výsledcích měření je možné pozorovat rychle se zvyšující průměrnou dobu nutnou pro přečtení snímku. Jediný výsledek, který by zajišťoval plynulé přehrávání výsledného videa je v rozlišení 320×180. Jedná se však o tak malé rozlišení, které by pro diváka nemělo dostatečný přínos. Z tohoto důvodu nebyla metoda využívající této funkce k nahrávání videa (viz 5.4.2.3) do aplikace Zoomera implementována. Naměřené časy potřebné pro přečtení snímku v rozlišení 320×180 popisuje histogram na Obrázku 74.

Rozlišení	**Průměrná doba čtení [ms]**	**Minimální doba čtení [ms]**	**Maximální doba čtení [ms]**	**Snímková frekvence [fps]**
320×180	32	26	48	**31.25**
640×360	104	93	124	**9.60**
1280×720	413	378	461	**2.42**
1920×1080	922	862	955	**1.08**

Tabulka 6. Výsledky testování funkce glReadPixels().

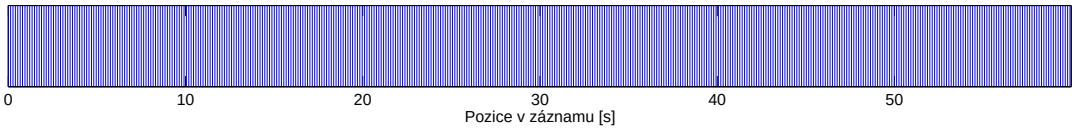
8.1.3. FFmpeg a kopie snímků

Testování metody nahrávání videa z kamery pomocí FFmpeg a kopie snímků (viz 5.4.2.2) je rozděleno na dvě části. První testuje, zda systém poskytuje kopie snímků s periodou dostatečnou pro provedení videozáznamu. Metoda získání kopie snímků je popsána v Kapitole 5.3.1. Druhá část se zabývá testováním samotného enkódování videa, kdy je měřena průměrná doba, která je potřeba pro zpracování jednoho snímku knihovnou FFmpeg (viz 5.4.1.2).

První měření bylo prováděno pro čtyři různá rozlišení kopie snímků (viz Tabulka 7). Princip je založen na změření průměrné periody, se kterou jsou ze systému získávány kopie snímků. Měření pro každé rozlišení probíhalo jednu minutu, během které byly zaznamenávány systémové časy získání snímku od systému. Poté byly spočítány rozdíly mezi po sobě následujícími snímky, ze kterých byla vypočtena výsledná průměrná hodnota. Z té byla určena odpovídající snímková frekvence.

Rozlišení	**Průměrná perioda výskytu snímku [ms]**	**Minimální perioda výskytu snímku [ms]**	**Maximální perioda výskytu snímku [ms]**	**Snímková frekvence [fps]**
320×240	34	21	42	**29.85**
640×480	34	15	56	**29.85**
1280×720	33	3	66	**29.85**
1920×1080	33	15	66	**29.85**

Tabulka 7. Výsledky měření odezvy systému při poskytování kopií snímků.



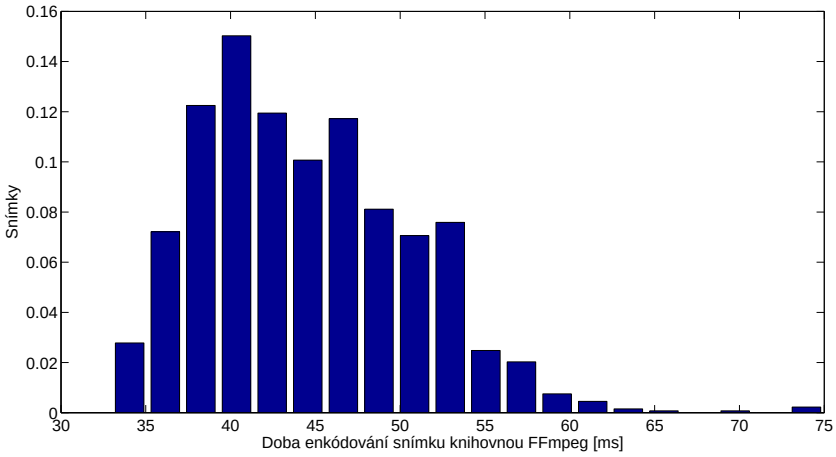
Obrázek 75. Znázornění časových značek kopií snímků v rozlišení 640×480. Je zobrazen každý čtvrtý snímek.

Jak dokazují výsledky měření, nehledě na rozlišení kopií snímků, byla výsledná snímková frekvence vždy stejná - 29.85 fps. Takový výsledek by zajisté poskytoval účinnou nahrávací metodu ve spojení s dostatečně rychlým enkódováním. Mimo jiné je tato metoda získání obrazových dat vhodná díky své rychlosti i pro použití s jinými algoritmy zpracovávání a rozpoznávání obrazu. Na Obrázku 75 jsou graficky znázorněny časové značky snímků získaných touto metodou. Časové značky netvoří shluky, jsou rovnoměrně rozložené po celé délce měření. To vypovídá o stálé frekvenci získávání kopií snímků bez jejich výpadků.

Video profil	**Průměrná doba enkódování snímku [ms]**	**Minimální doba enkódování snímku [ms]**	**Maximální doba enkódování snímku [ms]**	**Snímková frekvence [fps]**
320×240, 0.26 Mbps	11	4	37	**89.93**
640×480, 1.0 Mbps	45	33	75	**22.45**
1280×720, 4.1 Mbps	150	107	216	**6.65**
1920×1080, 7.1 Mbps	340	269	426	**2.94**

Tabulka 8. Výsledky měření průměrné doby potřebné k enkódování jednoho snímku knihovnou FFmpeg.

Druhé měření bylo prováděno během enkódování snímků pomocí knihovny FFmpeg pro čtyři rozlišení (viz Tabulka 8). Pro tyto účely byla použita aplikace FFmpegCamera implementována pro tuto práci (viz Kapitola 5.4.2.2 a Přiložené CD - Příloha A.3). Měřena byla doba potřebná pro enkódování jednoho snímku při daném rozlišení po dobu jedné minuty. Výsledkem byla průměrná doba potřebná pro enkódování jednoho snímku a tím byla známa i dosažitelná snímková frekvence.



Obrázek 76. Histogram naměřených časů potřebných pro enkódování snímku videa v rozlišení 640×480 pomocí knihovny FFmpeg.

Z výsledků je patrné, že pro rozlišení 320×240 a 640×480 by bylo možné enkódovat video z kamery tak, aby byl výsledek pro uživatele přínosem. Průměrná doba pro enkódování však se stoupajícím rozlišením rychle stoupá, a tak použití této metody ztrácí efektivitu. Naměřené časy potřebné enkování snímku v rozlišení 640×480 popisuje histogram na Obrázku 76.

8.1.4. Textura v OpenGL ES a MediaCodec

Testování nahrávání pomocí OpenGL ES textury a třídy *MediaCodec* bylo prováděno během tří měření, která se lišila množstvím aplikovaných efektů obrazu na výsledné video. První uvažovalo nahrávání videa z kamery bez jakýchkoli efektů, bez nastaveného digitálního zoom a se skrytou miniaturou. Druhé měření probíhalo s aplikovaným 10× zoom a se zobrazenou miniaturou. Třetí měření bylo prováděno s aplikovaným 10× zoom, se zobrazenou miniaturou a s aplikovanými efekty videa, jako je hodnota zoom, časová značka a logo (viz 5.6.1.2).

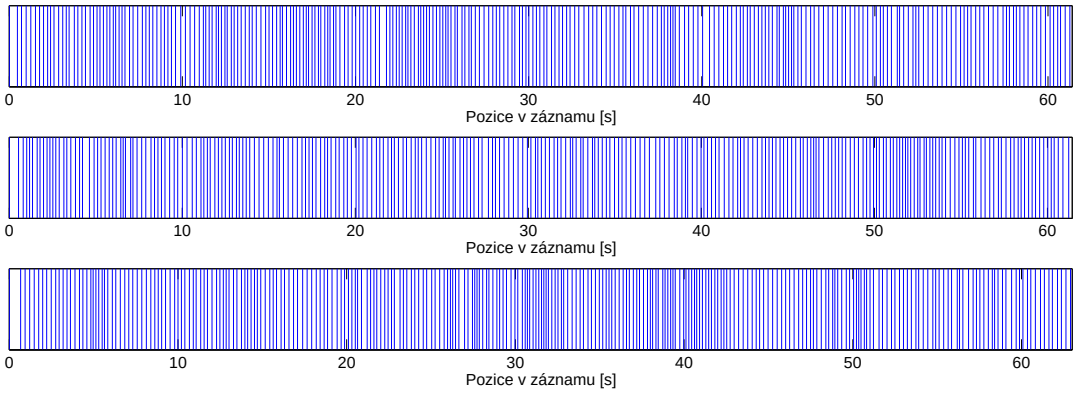
Všechna tři měření byla prováděna pro čtveřici rozlišení, kde pro každé rozlišení byl vytvořen záznam o délce jedné minuty. Nahrávání bylo prováděno v aplikaci Zoomera na zařízení LG Nexus 5. Výsledný záznam byl analyzován programem Avidemux[78], kde byla zjištěna délka záznamu, počet video snímků a průměrná snímková frekvence.

Video profil	**Počet snímků**	**Délka záznamu [m:s]**	**Snímková frekvence [fps]**
320×240, iF: 5 s, 0.95 Mbps	1197	1:03.039	**18.99**
640×480, iF: 5 s, 2.86 Mbps	1161	1:01.562	**18.86**
1280×720, iF: 5 s, 10.97 Mbps	1130	1:01.496	**18.38**
1920×1080, iF: 5 s, 15.26 Mbps	1107	1:01.462	**18.01**

Tabulka 9. Bez aplikovaných efektů.

Výsledky prvního měřeni bez aplikovaných efektů je v Tabulce 9. Změřené snímkové frekvence se pohybují v rozmezí od 18.01 - 18.99 fps a jejich hodnota klesá se stoupajícím rozlišením. Takto malý rozsah demonstruje, že změna rozlišení rozlišení zásadně neovlivňuje výslednou snímkovou frekvenci.

Na Obrázku 77 jsou graficky znázorněny časové značky snímků videí nahraných touto metodou. Je možné pozorovat, že časové značky tvoří shluky a nejsou rovnoměrně rozložené po celé délce záznamu. Vypovídá o proměnlivé snímkové frekvenci záznamu, během kterého docházelo k častým výpadkům snímků. To dokazuje výpočetní náročnost celého procesu, která je tak vyšší oproti systémovému nahrávání (viz Obrázek 72).



Obrázek 77. Znázornění časových značek snímků videa zaznamenaném nahráváním pomocí OpenGL ES textury a třídy MediaCodec v rozlišení 1920×1080. Je zobrazen každý čtvrtý snímek. Horní obrázek popisuje případ bez použití efektů, prostřední s aplikovaným zoom a zobrazenou miniaturou, dolní s aplikovaným zoom, zobrazenou miniaturou a efekty obrazu.

Pravděpodobně tedy samotná režie vykreslování má na nahrávání velký vliv, snížená snímková frekvence může být způsobena zdvojeným vykreslováním (viz 5.4.2.5), které

je během nahrávání výpočetně náročné. I tak je ale vytvořený záznam dostačující pro zachování plynulosti pohybu ve videu.

Video profil	**Počet snímků**	**Délka záznamu [m:s]**	**Snímková frekvence [fps]**
320×240, iF: 5 s, 0.95 Mbps	1212	1:01.660	**19.66**
640×480, iF: 5 s, 2.86 Mbps	1151	1:01.662	**18.67**
1280×720, iF: 5 s, 10.97 Mbps	1120	1:01.660	**18.16**
1920×1080, iF: 5 s, 15.26 Mbps	1096	1:01.493	**17.82**

Tabulka 10. Zobrazená miniatura obrazu a aplikovaný 10× zoom.

To, že na výslednou snímkovou frekvenci nahraného videa má velký vliv režie zdvojeného vykreslování potvrzují i výsledky z druhého (viz Tabulka 10) a třetího měření (viz Tabulka 11). I přes aplikaci různých obrazových efektů, které si vyžádají další volání grafické knihovny, není pozorovatelný větší pokles snímkové frekvence.

Video profil	**Počet snímků**	**Délka záznamu [m:s]**	**Snímková frekvence [fps]**
320×240, iF: 5 s, 0.95 Mbps	1209	1:01.762	**19.58**
640×480, iF: 5 s, 2.86 Mbps	1151	1:01.560	**18.70**
1280×720, iF: 5 s, 10.97 Mbps	1052	1:01.697	**17.05**
1920×1080, iF: 5 s, 15.26 Mbps	1142	1:03.104	**18.10**

Tabulka 11. Zobrazená miniatura obrazu, aplikovaný 10× zoom a povolené efekty videa (hodnota zoom, časová značka, logo)

8.2. Testování hlasového ovládání s uživateli

Pro účely ověření kvality rozpoznávání řeči ve spojení s ovládáním aplikace Zoomera touto metodou byl proveden test s vybraným vzorkem uživatelů. Vzorek uživatelů byl vybrán nezávisle na věku se stejným zastoupením mužů a žen. Protože jazykový a akustický model pro rozpoznávání je určen pro anglický jazyk, bylo požadováno, aby uživatelé ve vybraném vzorku měli alespoň základní znalost angličtiny. Bylo připraveno několik testovacích případů, které měly otestovat kvalitu detekce jednotlivých příkazů.

8.2.1. Použitá zařízení

LG Nexus 5 Mobilní telefon se systémem Android (viz Sekce 4.3.1).

Drátová sluchátka s mikrofonem Sluchátka dodávaná jako handsfree s mobilním telefonem Samsung Galaxy Nexus.

Bluetooth sluchátka Sony DR-BT21G (viz Obrázek 47).

8.2.2. Nastavení testů

Testování bylo nastaveno tak, aby uživatel postupně vyzkoušel všechny příkazy hlasového ovládání podporované aplikací Zoomera. Uživatel byl v uzavřené místnosti usazený u stolu, na kterém se nacházelo zařízení s nainstalovanou aplikací Zoomera (viz 8.2.1). Uživatel obdržel seznam možných příkazů, které měl provést. Během testování byl pod dohledem zapisovatele, který uživatele pozoroval a zaznamenával ke každému příkazu počet potřebných pokusů, než byl příkaz správně rozpoznán.

V první části se testovalo rozpoznávání se základním akustickým modelem. Uživatel postupně zadával hlasové příkazy, kdy zdrojem záznamu zvuku byl vestavěný mikrofon zařízení, uživatel byl nejprve v přibližně 30-50 cm vzdálenosti, následně ve vzdálenosti 1.5-2 m. Následně byl použit externí mikrofon na drátových sluchátkách určených k mobilnímu telefonu (viz 8.2.1) a Bluetooth sluchátka s mikrofonem (viz 8.2.1). Celý postup byl opakován s upraveným akustickým modelem pro konkrétního uživatele. Pokud nebyl některý z příkazů rozpoznán ani při jeho 5. opakování, byl označen za nerozpoznaný (počet pokusů zapsán jako x).

8.2.3. Testovací vzorek uživatelů

Testování bylo provedeno se čtyřmi uživateli ve věku od 20 do 60 let. Všichni uživatelé měli alespoň základní znalosti angličtiny a její výslovnosti.

- Uživatel 1 - Muž, 24 let
- Uživatel 2 - Žena, 20 let
- Uživatel 3 - Žena, 24 let
- Uživatel 4 - Muž, 60 let

8.2.4. Výsledky

8.2.4.1. Uživatel 1

Prvním uživatelem byl muž, věk 24 let. Při testování s vestavěným mikrofonem zařízení byla při malé vzdálenosti rozpoznána většina příkazů na první pokus. Rozpoznávání bylo úspěšné na několikátý pokus u příkazů *MY HIDE CONTROLS*, *MY SHOW CONTROLS* a *MY STOP RECORDING*, na druhý pokus u příkazu *MY MAXIMAL ZOOM*. Při umístění uživatele ve větší vzdálenosti od zařízení došlo k zhoršení rozpoznávání a některé příkazy nebyly rozpoznány vůbec.

Rozpoznávání s externím mikrofonem a Bluetooth mikrofonem proběhlo téměř bez jakékoli chyby. Jedinou výjimkou bylo 2. opakování příkazu *MY TAKE PHOTO* v případě externího mikrofonu.

Celý soubor testů byl zopakován i s upraveným akustickým modelem. Zde je možné pozorovat zlepšení u příkazů *MY HIDE CONTROLS*, *MY SHOW CONTROLS* a *MY STOP RECORDING*. Na druhou stranu došlo k zhoršení detekce u příkazu *MY MAXIMAL ZOOM* u vestavěného mikrofonu v malé vzdálenosti uživatele od zařízení.

Uživatel 1: Muž, 24 let

		POČET POKUSŮ							
		Standardní akustický model				Upravený akustický model			
	Fráze	Vestavěný mikrofon		Externí Mikrofon	Bluetooth Mikrofon	Vestavěný mikrofon		Externí Mikrofon	Bluetooth Mikrofon
		Blízko	Daleko			Blízko	Daleko		
1	MY ZOOM INCREASE	1	1	1	1	1	2	1	1
2	MY ZOOM DECREASE	1	1	1	1	1	1	1	1
3	MY MINIMAL ZOOM	1	x	1	1	2	x	1	1
4	MY MEDIUM ZOOM	1	1	1	1	1	1	1	1
5	MY MAXIMAL ZOOM	2	x	1	1	4	x	1	1
6	MY LIGHT ON	1	1	1	1	1	1	1	1
7	MY DARK	1	1	1	1	1	1	1	1
8	MY ENABLE AUTOFOCUS	1	1	1	1	1	1	1	1
9	MY DISABLE AUTOFOCUS	1	x	1	1	1	1	1	1
10	MY FOCUS	1	1	1	1	1	1	1	1
11	MY SHOW MINIATURE	1	2	1	1	1	1	1	1
12	MY HIDE MINIATURE	1	1	1	1	1	1	1	1
13	MY TAKE PHOTO	1	1	2	1	1	1	1	1
14	MY RETURN	1	1	1	1	1	1	1	1
15	MY SAVE PHOTO	1	1	1	1	1	1	1	1
16	MY HIDE CONTROLS	4	x	1	1	1	x	1	1
17	MY SHOW CONTROLS	5	5	1	1	1	4	1	1
18	MY START RECORDING	1	1	1	1	1	1	1	1
19	MY STOP RECORDING	3	1	1	1	1	1	1	1

Tabulka 12. Výsledky testování s uživatelem 1, Muž, 24 let

8.2.4.2. Uživatel 2

Uživatel 2: Žena, 20 let

		POČET POKUSŮ							
		Standardní akustický model				Upravený akustický model			
	Fráze	Vestavěný mikrofon		Externí Mikrofon	Bluetooth Mikrofon	Vestavěný mikrofon		Externí Mikrofon	Bluetooth Mikrofon
		Blízko	Daleko			Blízko	Daleko		
1	MY ZOOM INCREASE	1	1	1	1	1	1	1	1
2	MY ZOOM DECREASE	2	1	1	1	2	1	1	1
3	MY MINIMAL ZOOM	1	5	1	2	2	4	2	1
4	MY MEDIUM ZOOM	1	1	1	1	1	3	1	1
5	MY MAXIMAL ZOOM	x	x	1	1	x	x	1	1
6	MY LIGHT ON	1	1	1	1	1	1	1	1
7	MY DARK	1	1	1	1	1	1	1	1
8	MY ENABLE AUTOFOCUS	1	1	1	1	1	1	1	1
9	MY DISABLE AUTOFOCUS	x	x	1	5	x	x	2	x
10	MY FOCUS	1	1	1	1	1	4	1	1
11	MY SHOW MINIATURE	2	1	1	1	1	1	1	1
12	MY HIDE MINIATURE	1	1	1	1	1	3	1	1
13	MY TAKE PHOTO	1	1	1	1	1	3	1	1
14	MY RETURN	1	3	1	1	3	x	1	2
15	MY SAVE PHOTO	1	1	1	1	2	1	1	1
16	MY HIDE CONTROLS	1	x	1	1	1	x	1	1
17	MY SHOW CONTROLS	1	x	2	1	2	x	1	1
18	MY START RECORDING	1	5	1	1	1	x	1	3
19	MY STOP RECORDING	3	3	1	3	5	2	1	1

Tabulka 13. Výsledky testování s uživatelem 2, Žena, 20 let

Druhým uživatelem byla žena, věk 20 let. Při testování s vestavěným mikrofonem zařízení byla při malé vzdálenosti rozpoznána většina příkazů na první pokus nebo druhý pokus. Objevovaly se však příkazy, které nebyly rozpoznány vůbec - *MY MAXIMAL ZOOM* a *MY DISABLE AUTOFOCUS*. Při větší vzdálenosti uživatele od zařízení se úspěšnost detekce ještě zhoršila a nebyly tak rozpoznány navíc příkazy *MY HIDE CONTROLS* a *MY SHOW CONTROLS*.

Rozpoznávání s externím mikrofonem proběhlo téměř bez jakékoli chyby. Jedinou výjimkou bylo 2. opakování příkazu *MY SHOW CONTROLS*. V případě Bluetooth mikrofonu byly výsledky podobné, příkaz *MY DISABLE AUTOFOCUS* se podařilo rozpoznat až na 5. pokus.

Celý soubor testů byl zopakován i s upraveným akustickým modelem. Zde bohužel nebylo možné pozorovat zlepšení mezi upraveným a neupraveným modelem. Rozpoznání se u některých příkazů zlepšilo, u některých zhoršilo.

8.2.4.3. Uživatel 3

Třetím uživatelem byla žena, věk 24 let. Při testování s vestavěným mikrofonem zařízení byla při malé vzdálenosti rozpoznána většina příkazů na první nebo druhý pokus. Objevovaly se však příkazy, které nebyly rozpoznány vůbec - *MY MAXIMAL ZOOM* a *MY SHOW CONTROLS*. Při větší vzdálenosti uživatele od zařízení se úspěšnost detekce zhoršila a nebyly tak rozpoznány navíc příkazy *MY ZOOM DECREASE*, *MY MINIMAL ZOOM*, *MY HIDE MINIATURE*, *MY HIDE CONTROLS* a *MY START RECORDING*.

Rozpoznávání s externím a Bluetooth mikrofonem proběhlo téměř bez jakékoli chyby. Jedinou výjimkou bylo opakování příkazů *MY MINIMAL ZOOM* a *MY START RECORDING*.

Uživatel 3: Žena, 24 let

		POČET POKUSŮ							
		Standardní akustický model				Upravený akustický model			
	Fráze	Vestavěný mikrofon		Externí Mikrofon	Bluetooth Mikrofon	Vestavěný mikrofon		Externí Mikrofon	Bluetooth Mikrofon
		Blízko	Daleko			Blízko	Daleko		
1	MY ZOOM INCREASE	2	1	2	1	2	2	1	4
2	MY ZOOM DECREASE	1	x	1	1	1	1	1	1
3	MY MINIMAL ZOOM	2	x	2	3	4	3	1	1
4	MY MEDIUM ZOOM	1	4	1	1	1	x	1	1
5	MY MAXIMAL ZOOM	x	x	1	1	1	2	1	2
6	MY LIGHT ON	2	2	1	1	1	x	1	1
7	MY DARK	1	2	1	1	3	1	1	2
8	MY ENABLE AUTOFOCUS	1	3	1	1	1	2	1	1
9	MY DISABLE AUTOFOCUS	2	2	1	1	3	1	1	2
10	MY FOCUS	5	1	1	1	4	1	1	1
11	MY SHOW MINIATURE	1	1	1	1	1	1	1	1
12	MY HIDE MINIATURE	1	x	1	1	1	2	1	1
13	MY TAKE PHOTO	1	1	1	1	1	1	1	1
14	MY RETURN	2	4	1	1	2	3	1	1
15	MY SAVE PHOTO	5	2	1	1	1	3	1	1
16	MY HIDE CONTROLS	1	x	1	1	4	x	1	3
17	MY SHOW CONTROLS	x	x	1	1	4	x	1	2
18	MY START RECORDING	5	x	4	3	2	x	1	3
19	MY STOP RECORDING	1	1	1	1	1	1	1	1

Tabulka 14. Výsledky testování s uživatelem 3, Žena, 24 let

Celý soubor testů byl zopakován i s upraveným akustickým modelem. Zde došlo k mírnému zlepšení takovému, že v případech, kdy u neupraveného modelu nebyly některé příkazy rozpoznány vůbec, byly u upraveného modelu úspěšně rozpoznány. I tak ale bylo u většiny případů nutné příkazy opakovat. Je možné pozorovat velký rozdíl mezi externím mikrofonem a Bluetooth mikrofonem. Jeden z možných důvodů může být ten, že přenos zvuku přes Bluetooth je komprimován a může tak docházet ke ztrátě kvality audia v porovnání s drátovými sluchátky.

8.2.4.4. Uživatel 4

Čtvrtým uživatelem byl muž, věk 60 let. Je důležité zmínit, že uživatel měl horší anglickou výslovnost, která velmi zásadně ovlivnila výsledek testu. Při testování s vestavěným mikrofonem zařízení bylo při malé vzdálenosti nutno velkou část příkazů několikrát

opakovat a některé příkazy nebyly rozpoznány vůbec - *MY MINIMAL ZOOM*, *MY MEDIUM ZOOM*, *MY MAXIMAL ZOOM*, *MY DISABLE AUTOFOCUS* a *MY SHOW CONTROLS*. Při větší vzdálenosti uživatele od zařízení se úspěšnost detekce zhoršila a nebyly tak rozpoznány navíc příkazy *MY ENABLE AUTOFOCUS*, *MY TAKE PHOTO*, *MY RETURN*, *MY HIDE CONTROLS* a *MY STOP RECORDING*.

Rozpoznávání s externím mikrofonem proběhlo s menším počtem chyb, ale příkaz *MY MINIMAL ZOOM* nebyl rozpoznán vůbec. V případě použití Bluetooth mikrofonu byla úspěšnost podobná, ale navíc nebyly rozpoznány příkazy *MY RETURN*, *MY HIDE CONTROLS* a *MY SHOW CONTROLS*.

Celý soubor testů byl zopakován i s upraveným akustickým modelem. Zde došlo k mírnému zlepšení u vestavěného mikrofonu. Oproti tomu je viditelné zlepšení u Bluetooth a u externího mikrofonu, kde proběhlo rozpoznávání téměř bez chyby.

Testování bylo velmi ovlivněno horší výslovností uživatele, ale dá se pozorovat, že díky úpravě akustického modelu je možné zlepšit rozpoznání mluvčího se zhoršenou kvalitou výslovnosti.

Uživatel 4: Muž, 60 let

		POČET POKUSŮ							
		Standardní akustický model				Upravený akustický model			
	Fráze	Vestavěný mikrofon		Externí	Bluetooth	Vestavěný mikrofon		Externí	Bluetooth
		Blízko	Daleko	Mikrofon	Mikrofon	Blízko	Daleko	Mikrofon	Mikrofon
1	MY ZOOM INCREASE	1	1	1	1	1	x	1	1
2	MY ZOOM DECREASE	3	5	1	1	1	1	1	1
3	MY MINIMAL ZOOM	x	x	x	x	x	x	1	1
4	MY MEDIUM ZOOM	x	3	1	3	x	1	1	1
5	MY MAXIMAL ZOOM	x	x	3	5	x	x	3	1
6	MY LIGHT ON	3	1	1	1	2	1	1	2
7	MY DARK	1	1	1	1	1	1	1	1
8	MY ENABLE AUTOFOCUS	4	x	1	1	3	x	1	1
9	MY DISABLE AUTOFOCUS	x	3	1	1	4	2	1	1
10	MY FOCUS	1	1	1	2	1	4	1	1
11	MY SHOW MINIATURE	1	3	1	1	3	4	1	1
12	MY HIDE MINIATURE	1	1	1	1	3	1	1	1
13	MY TAKE PHOTO	5	x	1	1	3	x	1	1
14	MY RETURN	1	x	2	x	1	3	1	x
15	MY SAVE PHOTO	1	3	1	1	2	1	1	1
16	MY HIDE CONTROLS	3	x	1	x	x	x	1	3
17	MY SHOW CONTROLS	x	x	2	x	x	x	1	x
18	MY START RECORDING	3	2	1	1	1	4	1	1
19	MY STOP RECORDING	2	x	1	3	3	x	1	2

Tabulka 15. Výsledky testování s uživatelem 4, Muž, 60 let

8.2.4.5. Shrnutí

U všech uživatelů byl pozorován velmi velký rozdíl v kvalitě detekce při použití vestavěného mikrofonu a externích mikrofonů. Přesnost detekce se při použití externích mikrofonů výrazně zlepšila a vede to tak k závěru, že mikrofon vestavěný v zařízení neposkytuje dostatečnou kvalitu zaznamenávaného zvuku. Bylo by vhodné provést testování i na jiných zařízeních, aby bylo zjištěno, zda se problém špatné kvality vestavěného mikrofonu týká pouze použitého zařízení, či mobilních zařízení vůbec.

Úprava akustického modelu přinesla ve všech případech jen mírné zlepšení detekce, v některých případech se detekce dokonce zhoršila. To může být zapříčiněno tím, že byla zvolena špatná množina ukázkových vzorků určených k trénování akustického modelu, nebo bylo vzorků příliš málo. Možností pro budoucí rozšíření je vytvoření velké databáze ukázkových vzorků hlasu mnoha uživatelů a vytvoření vlastního akustického modelu.

U uživatele 4 bylo zjištěno, že úprava akustického modelu může zlepšit rozpoznávání řeči v případě, že má uživatel horší anglickou výslovnost. V kombinaci s externím mikrofonem bylo pozorováno znatelné zlepšení rozpoznávání u upraveného akustického modelu. Model byl upraven vzorky nahranými na testovacím zařízení s uživatelem 4.

Docházelo k problémům s rozpoznáváním příkazů *MY SHOW CONTROLS*, *MY HIDE CONTROLS*, *MY MINIMAL ZOOM*, *MY MAXIMAL ZOOM* a *MY DISABLE AUTOFOCUS*. Nejedná se o příkazy, u kterých se předpokládá časté použití a nebylo by vhodné je měnit. Mohlo by to ovlivnit ty, které fungují a budou používány častěji.

Uživatelé často uváděli, že některé příkazy nejsou vhodně zvoleny, ať už z důvodu délky, nebo velké složitosti. Jednalo se převážně o příkazy, které by uživatelé podle svého názoru nejčastěji používali. A to zejména *MY ZOOM INCREASE*, *MY ZOOM DECREASE*, *MY LIGHT ON*, *MY TAKE PHOTO* a *MY SAVE PHOTO*.

8.2.5. Předefinování příkazů vzhledem k výsledkům testů

Testování odhalilo příkazy, které uživatelé považovali za nevhodné pro svou délku nebo velkou složitost. Z toho důvodu byla provedena redefinice uživateli nejčastěji zmiňovaných příkazů. Některé z příkazů byly zcela nahrazeny novými (viz Tabulka 16) a některé byly ponechány a doplněny o alternativy (viz Tabulka 17). Byly také vytvořeny příkazy zcela nové (viz Tabulka 18).

Původní příkaz	**Nový příkaz**
MY LIGHT ON	MY LIGHT
MY TAKE PHOTO	MY SNAP
MY SAVE PHOTO	MY SAVE

Tabulka 16. Změněné příkazy.

Původní příkaz	**Vytvořená alternativa**
MY ZOOM INCREASE	MY ZOOM IN
MY ZOOM DECREASE	MY ZOOM OUT

Tabulka 17. Nově vytvořené alternativní příkazy.

Nový příkaz	**Funkce**
MY ZOOM <číslo od 1 do 20>	Sada příkazů umožňuje nastavení celočíselného koeficientu zvětšení od 1 do 20.

Tabulka 18. Nové příkazy.

Příkaz pro rozsvícení diody *MY LIGHT ON* obsahoval tři slova, i když mezi ostatními příkazy nebyl žádný, který by se mu podobal a mohlo tak dojít k záměně. Byl tak nahrazen příkazem *MY LIGHT*, který je tak konzistentní s příkazem *MY DARK*, který diodu zhasíná. Podobně byly nahrazeny i příkazy *MY TAKE PHOTO* a *MY SAVE PHOTO*, vznikly tak nové a kratší příkazy *MY SNAP* a *MY SAVE*.

Ovládání digitálního zoom pomocí příkazů *MY ZOOM INCREASE* a *MY ZOOM DECREASE* bylo rozšířeno o alternativní příkazy *MY ZOOM IN* a *MY ZOOM OUT*, které používají často se vyskytující fráze ve spojení s přibližováním a oddalováním obrazu.

Přidána byla sada příkazů umožňující nastavit konkrétní celočíselnou hodnotu digitálního zoom. Aktivace je provedena příkazem MY ZOOM <číslo od 1 do 20>. Čísla jsou v anglickém jazyce, jeden z příkazů tedy může vypadat následovně: MY ZOOM TWELVE. Přidáním nových příkazů byl také zásadně rozšířen slovník o dvacet jednotlivých číslic. Taková změna slovníku se může negativně projevit na přesnosti detekce.

8.3. Výpočetní a paměťové nároky rozpoznávacího modulu

8.3.1. Použité zařízení

Pro měření bylo použito zařízení LG Nexus 5 (viz Kapitola 4.3.1).

8.3.2. Způsob měření výpočetních a paměťových nároků

Výpočetní a paměťové nároky byly měřeny za pomocí vývojářských nástrojů z Android SDK verze 22.0.5 v vývojovém prostředí Eclipse verze 3.8. Byl použit nástroj Threads pro sledování system a user time jednotlivých vláken aplikace, nástroj System information pro pozorování zatížení CPU a nástroj Heap pro sledování alokace Java Heap aplikace Zoomera.

Aplikace byla sledována ve dvou stavech. V klidu, kdy bylo spuštěno pouze vykreslování snímků z kamery na obrazovku a se zapnutým rozpoznáváním řeči. Ve obou případech po dobu deseti minut. Všechny hodnoty velikosti paměti byly zaokrouhleny na desetiny MB, procentuální hodnoty byly zaokrouhleny s přesností na 0.5%, počty objektů byly zaokrouhleny s přesností na 100.

8.3.2.1. System time a User time u vláken

Nástrojem Threads byly měřeny časy vláken aplikace Zoomera. Byly měřeny dva druhy času - prvním je system time, čas, který stráví dané vlákno výpočtem na procesoru v jádru systému. Druhým časem je user time, jedná se o čas, který stráví vlákno výpočtem na procesoru mimo jádro systému.

8.3.2.2. Procentuální využití času procesoru

Nástroj System information umožňuje sledování vytížení procesoru zařízení jednotlivými spuštěnými procesy. Aplikace Zoomera využívá svůj vlastní proces běžící v user prostoru a v system prostoru. Dále však využívá i další procesy pro ovládání kamery zařízení a dalších periférií a také proces, který zajišťuje vykreslování obrazovky.

8.3.2.3. Velikost Java Heap

Aplikace Zoomera je spuštěna v Dalvik Virtual Machine, což je Java Virtual Machine určená pro systém Android. Jako každá Java aplikace má i tato Java Heap, která slouží jako paměťový prostor pro dynamickou alokaci objektů. Nástroj Heap umožňuje její detailní sledování pro aktuálně spuštěný proces.

8.3.3. Výpočetní nároky

Výpočetní nároky byly měřeny jako procentuální využití času procesoru a System time a User time u vláken. Měřeny byly dva případy: aplikace v klidu, kdy bylo spuštěno pouze vykreslování snímků z kamery na obrazovku a se zapnutým rozpoznáváním řeči.

8.3.3.1. Klidový stav

Vytížení procesoru aplikací Zoomera se během měření pohybovalo v rozmezí od 35% do 40%.

Vlákno zajišťující rozpoznávání řeči nevyžadovalo žádné prostředky. Vlákno překreslování obrazovky vyžadovalo za dobu běhu měření 72% user time a 42% system time.

Celkově vyžadovalo vlákno 59% celkového času procesoru přidělenému aplikaci. Vlákno určené pro získávání debugovacích informací vyžadovalo za dobu běhu měření 10% user time a 38% system time. Celkově vyžadovalo vlákno 22% celkového času procesoru přidělenému aplikaci.

Ostatní vlákna vyžadovala za dobu běhu měření 18% user time a 20% system time. Celkově vyžadovala ostatní vlákna 19% celkového času procesoru přidělenému aplikaci.

8.3.3.2. Spuštěné hlasové ovládání

Vytížení procesoru aplikací Zoomera se během měření pohybovalo v rozmezí od 45% do 60%.

Vlákno zajišťující rozpoznávání řeči vyžadovalo za dobu běhu měření 63% user time a 2,5% system time. Celkově vyžadovalo vlákno 43% celkového času procesoru přidělenému aplikaci. Vlákno překreslování obrazovky vyžadovalo za dobu běhu měření 19% user time a 17,5% system time. Celkově vyžadovalo vlákno 18.5% celkového času procesoru přidělenému aplikaci. Vlákno určené pro získávání debugovacích informací vyžadovalo za dobu běhu měření 14% user time a 72% system time. Celkově vyžadovalo vlákno 33% celkového času procesoru přidělenému aplikaci.

Ostatní vlákna vyžadovala za dobu běhu měření 4% user time a 8% system time. Celkově vyžadovala ostatní vlákna 5,5% celkového času procesoru přidělenému aplikaci.

8.3.3.3. Shrnutí

Naměřené hodnoty vytížení procesoru i časy vláken dokazují, že spuštěné rozpoznávání řeči vyžaduje nezanedbatelné množství výpočetních prostředků. Systém nedosahoval kritických hodnot zatížení, ale modul rozpoznávání řeči má zcela jistě vliv na rychlejší vybíjení baterie. V případech, kdy by bylo zařízení vytíženo i jinými procesy, než je aplikace Zoomera, mohlo by dojít k zhoršení reakční doby systému vlivem nedostatku výpočetních prostředků.

8.3.4. Paměťové nároky

Paměťové nároky byly měřeny vzhledem k velikosti a alokaci Java Heap aplikace. Měřeny byly dva případy: aplikace v klidu, kdy bylo spuštěno pouze vykreslování snímků z kamery na obrazovku a se zapnutým rozpoznáváním řeči.

8.3.4.1. Klidový stav

Velikost Java Heap aplikace byla během měření 41,5 MB. Z toho bylo alokováno v rozmezí od 40,0 do 40,5 MB. Průměrný počet alokovaných objektů byl 50500.

8.3.4.2. Spuštěné hlasové ovládání

Velikost Java Heap aplikace byla během měření 51,5 MB. Z toho bylo alokováno v rozmezí od 43,0 do 43,5 MB. Průměrný počet alokovaných objektů byl 51000.

8.3.4.3. Shrnutí

Spuštěné hlasové ovládání zvýšilo velikost Java Heap o 10 MB, ale velikost alokovaného prostoru se zvýšila jen o 3 MB. Takto malý rozdíl není pro běh systému nijak omezující.

9. Závěr

Úkolem této práce bylo upravit existující aplikaci Zoomera 2012 takovým způsobem, aby sloužila jako pokročilý záznamník videa a umožňovala ovládání hlasovými povely při kontinuálním rozpoznávání řeči. Hlavní náplní byla implementace potřebných modulů, dále pak experimenty, testování a příslušná úprava grafického uživatelského rozhraní.

Výsledkem práce je nástroj, který se zcela liší od dostupných kamerových aplikací. Hlavní jeho předností je možnost provádět pokročilý záznam videa z kamery zařízení, s úpravou obrazu na grafickém čipu zařízení a kompresí videa pomocí systémových kodeků. Přidanou hodnotou je možnost jednotlivé snímky zpracovávat v reálném čase ještě před uložením do videa. Díky tomu je možné nahrávat obraz s aplikovanými filtry a přidanými informacemi, jako je časová značka, hodnota zvětšení obrazu nebo vložení statického obrázku.

Funkčnost záznamu je dále rozšířena o nahrávání v nekonečné smyčce, což umožňuje užití aplikace při pozorování a kontrole nejrůznějších činností. To dovoluje uživateli dlouhodobě zaznamenávat pozorovanou scénu, kdy je důležité mít přístup k záznamu pořízeného v posledních několika časových jednotkách. Uvažovaným příkladem použití je zaznamenávání činnosti pracovníka laboratoře za účelem kontroly kvality. Nahrávání ve smyčce umožní pracovníkovi provést inspekci provedených úkonů provedených v posledních okamžicích. Díky této funkci nedochází k zaplnění paměti zařízení celým dlouhým záznamem, ale vždy jen velikostí nahrávané smyčky. Záznam je tak možné provádět nepřetržitě.

Možnosti použití rozšiřuje hlasové ovládání, které dává uživateli volnost tím, že nemusí být v těsném kontaktu se zařízením. Je uvažováno použití v laboratořích, ať už z důvodu nemožnosti ovládání aplikace dotykem, protože se uživatel věnuje jiné činnosti, nebo protože se nachází v kontaminovaném prostředí. Byla vyhodnocena řada open source nástrojů pro rozpoznávání řeči dostupných na trhu. Z nich byla pro použití v aplikaci Zoomera vybrána knihovna Pocketsphinx z projektu CMU Sphinx, která jako jediná umožňovala plnou kontrolu nad procesem rozpoznávání, umožňovala tvorbu a úpravu jazykových modelů a byla k dispozici pro systém Android. Díky tomu bylo možné vytvořit v aplikaci hlasové ovládání, které rozšiřuje výše zmíněné možnosti použití aplikace.

Robustní řešení pro ovládání aplikace hlasem vzniká v kombinaci s drátovými sluchátky nebo Bluetooth headsetem, který zaznamenává převážně hlas uživatele a rušivé zvuky okolí tak nemají vliv na detekci. Spuštění hlasového ovládání zastoupí jakékoli jiné uživatelské vstupy, a je možné tak aplikaci ovládat jen za pomocí řeči uživatele.

Žádná z uvedených funkčností není obsažena v běžně používaných systémech na záznam videa dostupných na Google Play. Vznikl tak nástroj s vlastnostmi, které nezastoupí žádná jiná aplikace určená pro zaznamenávání videa. Je obsažena implementace nestandardních funkcí, pro které chybí detailní popis použití v dokumentaci Androidu. Tato práce tyto chybějící části pokrývá a může sloužit jako zdroj informací při vývoji nahrávání videozáznamu na platformě Android.

Byly prozkoumány metody nahrávání videa z kamery v systému Android, jejich výhody a nevýhody. Uvažováno bylo jen nahrávání videa bez zvuku, protože za současného použití hlasového vstupu by bylo obtížně až nemožné metody implementovat. Byla

provedena měření, jejichž výsledky demonstrují efektivitu použití jednotlivých metod a jejich součástí na konkrétním zařízení. Dvě z těchto metod byly do aplikace Zoomera implementovány. První je metoda využívající systémové funkce, má svou výhodu v kvalitním záznamu, jehož zpracování systémem je dobře výpočetně optimalizováno. Druhá zpracovává obraz na GPU a využívá systémových kodeků. Ačkoliv z výsledků měření vyplynulo, že oproti první uvedené metodě je tato výpočetně náročnější, přínosem je možnost úpravy nahrávaného obrazu. Protože jsou použity nové systémové kodeky a třídy pro práci s multimédii, vyžaduje implementovaná aplikace Android verze 4.3 a vyšší.

Implementace nahrávání ve smyčce je provedena pomocí uložení posledního časového intervalu do dočasných souborů, které jsou po dobu nahrávání vytvářeny a mazány v závislosti na stáří snímků, které obsahují. Během nahrávání je v zařízení uložen pouze záznam obsahující aktuální část smyčky a po dokončení je vytvořen video soubor obsahující smyčku.

Hlasové ovládání je založeno na použití aktivačního slova spolu s požadovaným příkazem. Nástroje z projektu CMU Sphinx posloužily k testování možnosti úpravy akustického modelu. Pro nahrání zvukových ukázek hlasu potřebných k úpravě akustického modelu byla vytvořena aplikace, která provedení tohoto kroku zjednodušila. Možnost použití bezdrátových sluchátek byla zajištěna podporou vytvoření Bluetooth audio tunelu.

Testováním hlasového ovládání s uživateli bylo zjištěno, že rozpoznávání při použití vestavěného mikrofonu zařízení dosahuje zvýšené chybovosti, oproti tomu rozpoznávání s použitím externího mikrofonu se dopouští jen málo chyb. Při použití upraveného akustického modelu pro konkrétního uživatele bylo pozorováno v některých případech mírné zlepšení, v některých však zhoršení rozpoznávání řeči. V případě jednoho uživatele, který měl horší anglickou výslovnost, bylo pozorováno znatelné zlepšení při použití adaptovaného akustického modelu. To může naznačovat skutečnost, že by úprava akustického modelu mohla v podobných případech pomoci zlepšit rozpoznávání řeči.

Zvuky okolního prostředí mají zásadní vliv na funkčnost rozpoznávacího modulu. Předpokládá se tak, že bude hlasové ovládání používáno v kombinaci s drátovými sluchátky nebo Bluetooth headsetem. V opačném případě je doporučeno použití v místech, kde je snížený okolní hluk prostředí. Rušivé zvuky ve spojení s příkazy hlasového ovládání jsou jen těžko rozpoznatelné a aplikaci by nebylo možné pohodlně ovládat.

Budoucí možné rozšíření by se mohlo zaměřit na zdokonalení stávajících a přidání nových možností úprav obrazu v reálném čase. Dále by se mohlo týkat testování použitelnosti aplikace jak z hlediska uživatelského rozhraní, tak z hlediska použitých hlasových příkazů. Testování s větším počtem uživatelů by mohlo poskytnout nové možnosti nastavení hlasového ovládání a další zdokonalení aplikace.

Příloha A.

A.1. Skript pro úpravu akustického modelu

```
rm -R hub4wsj_sc_8kadapt

sphinx_fe  -argfile hub4wsj_sc_8k/feat.params -samprate 16000 -di .
           -do . -c zoomera.listoffiles -ei wav -eo mfc -mswav yes

pocketsphinx_mdef_convert -text hub4wsj_sc_8k/mdef hub4wsj_sc_8k/mdef.txt

./bw \
    -hmmdir hub4wsj_sc_8k \
    -moddeffn hub4wsj_sc_8k/mdef.txt \
    -ts2cbfn .semi. \
    -feat 1s_c_d_dd \
    -svspec 0-12/13-25/26-38 \
    -cmn current \
    -agc none \
    -dictfn zoomera.dic \
    -ctlfn zoomera.listoffiles \
    -lsnfn zoomera.transcription \
    -accumdir . \
    -cmninit 56,-3,1 \
    -varnorm no

/usr/local/libexec/sphinxtrain/mllr_solve \
    -meanfn hub4wsj_sc_8k/means \
    -varfn hub4wsj_sc_8k/variances \
    -outmllrfn mllr_matrix -accumdir .

cp -a hub4wsj_sc_8k hub4wsj_sc_8kadapt

./map_adapt \
    -meanfn hub4wsj_sc_8k/means \
    -varfn hub4wsj_sc_8k/variances \
    -mixwfn hub4wsj_sc_8k/mixture_weights \
    -tmatfn hub4wsj_sc_8k/transition_matrices \
    -accumdir . \
    -mapmeanfn hub4wsj_sc_8kadapt/means \
    -mapvarfn hub4wsj_sc_8kadapt/variances \
    -mapmixwfn hub4wsj_sc_8kadapt/mixture_weights \
    -maptmatfn hub4wsj_sc_8kadapt/transition_matrices

./mk_s2sendump \
    -pocketsphinx yes \
    -moddeffn hub4wsj_sc_8kadapt/mdef.txt \
    -mixwfn hub4wsj_sc_8kadapt/mixture_weights \
    -sendumpfn hub4wsj_sc_8kadapt/sendump

rm hub4wsj_sc_8kadapt/mixture_weights
rm hub4wsj_sc_8kadapt/mdef.txt
```

Obrázek 78. Skript umožňující adaptaci akustického modelu na základě nástrojů CMU Sphinx a poskytnutých zvukových ukázek.

A.2. Zápis hlavičky WAVE souboru

```
1  private static void writeWAVHeader(FileChannel fc, int sampleRateInHz,
2                                     short numChannels, int
                                          sizeOfPCMInBytes) {
3      byte[] chunkID     = { 'R', 'I', 'F', 'F' };
4      byte[] chunkFormat = { 'W', 'A', 'V', 'E' };
5      byte[] subChunk1ID = { 'f', 'm', 't', ' ' };
6      byte[] subChunk2ID = { 'd', 'a', 't', 'a' };
7
8      int subChunk1Size = 16;
9      int subChunk2Size = sizeOfPCMInBytes;
10     int chunkSize = 4 + (8 + subChunk1Size) + (8 + subChunk2Size);
11
12     short audioFormatHeader = 1;
13     short bitsPerSample = 16;
14     int byteRate = sampleRateInHz * (int) numChannels * (int) bitsPerSample
           / 8;
15     short blockAlign = (short) (numChannels * bitsPerSample / 8);
16
17     // The RIFF chunk descriptor
18     FileOperations.writeBufferToFile(chunkID, fc, ByteOrder.BIG_ENDIAN);
19     FileOperations.writeIntToFile(chunkSize, fc, ByteOrder.LITTLE_ENDIAN);
20     FileOperations.writeBufferToFile(chunkFormat, fc, ByteOrder.BIG_ENDIAN);
21     // The fmt sub-chunk
22     FileOperations.writeBufferToFile(subChunk1ID, fc, ByteOrder.BIG_ENDIAN);
23     FileOperations.writeIntToFile(subChunk1Size, fc,
           ByteOrder.LITTLE_ENDIAN);
24     FileOperations.writeShortToFile(audioFormatHeader, fc,
           ByteOrder.LITTLE_ENDIAN);
25     FileOperations.writeShortToFile(numChannels, fc,
           ByteOrder.LITTLE_ENDIAN);
26     FileOperations.writeIntToFile(sampleRateInHz, fc,
           ByteOrder.LITTLE_ENDIAN);
27     FileOperations.writeIntToFile(byteRate, fc, ByteOrder.LITTLE_ENDIAN);
28     FileOperations.writeShortToFile(blockAlign, fc,
           ByteOrder.LITTLE_ENDIAN);
29     FileOperations.writeShortToFile(bitsPerSample, fc,
           ByteOrder.LITTLE_ENDIAN);
30     // The data sub-chunk
31     FileOperations.writeBufferToFile(subChunk2ID, fc, ByteOrder.BIG_ENDIAN);
32     FileOperations.writeIntToFile(subChunk2Size, fc,
           ByteOrder.LITTLE_ENDIAN);
33 }
```

Obrázek 79. Metoda *writeWAVHeader()*, která zapíše do souboru hlavičku WAVE formátu na základě vstupních parametrů metody.

A.3. Obsah přiloženého CD

```
/
├── lib                                  Použité knihovny
│   ├── CMUSphinx                        Knihovny pro rozpoznávání řeči
│   └── FFmpeg                           Knihovna pro práci s multimédii
├── prace                                Tato práce
│   ├── latex                            Text ve zdrojovém formátu
│   └── buzektom-dp-2014.pdf
├── src                                  Zdrojové kódy aplikací
│   ├── adapt-acoustic-model             Adaptace akustického modelu
│   ├── AudioRecorder                    Nahrávání hlasových vzorků
│   │   ├── assets
│   │   ├── res
│   │   ├── src
│   │   └── AndroidManifest.xml
│   ├── FFmpegCamera                     Záznam videa z kamery s FFmpeg
│   │   ├── jni
│   │   ├── libs
│   │   ├── res
│   │   ├── src
│   │   └── AndroidManifest.xml
│   ├── jazykovy-model                   Soubory pro hlasové ovládání
│   ├── Zoomera                          Aplikace Zoomera
│   │   ├── assets
│   │   ├── jni
│   │   ├── libs
│   │   ├── res
│   │   ├── src
│   │   └── AndroidManifest.xml
└── video                                Videa nahraná aplikací Zoomera
    ├── edge-detection.mp4
    ├── video-30-04-2014_19-48-43.mp4
    ├── video-30-04-2014_19-50-49.mp4
    └── video-effects.mp4
```

Literatura

[1] Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store.

[2] Fotoaparát Google - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=com.google.android.GoogleCamera.

[3] Candy Camera - Selfie Camera - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=com.joeware.android.gpulumera.

[4] Fotoaparát ZOOM FX - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=slide.cameraZoom.

[5] Camera MX - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=com.magix.camera_mx.

[6] Paper Camera - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=com.dama.papercamera.

[7] A Better Camera - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=com.almalence.opencam.

[8] Zoom Camera - Aplikace pro Android ve službě Google Play. [online]. [cit. 2014-05-09]. Dostupné z: https://play.google.com/store/apps/details?id=ar.com.moula.zoomcamera.

[9] Tomáš Buzek. *Aplikace "lupa" pro operační systém Android*. Bakalářská práce. Praha, 2012.

[10] OpenGL ES - The Standard for Embedded Accelerated 3D Graphics. Khronos Group. [online]. [cit. 2014-01-28]. Dostupné z: http://www.khronos.org/opengles/.

[11] Nexus 5 - D821 - Android 4.4 KitKat - Mobilní telefony - LGE CZ. [online]. [cit. 2014-02-02]. Dostupné z: http://www.lg.com/cz/telefony/lg-D821-nexus-5.

[12] I.E. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley, 2011. ISBN: 9781119965305.

[13] A. Beach. *Real World Video Compression*. Pearson Education, 2010 . ISBN: 9780132089517.

[14] K. Jack. *Video Demystified: A Handbook for the Digital Engineer*. Video Demystified Series. Elsevier Science, 2011. ISBN: 9780080553955.

[15] J.D. Foley et al. *Computer Graphics: Principles and Practice*. The systems programming series. ADDISON WESLEY Publishing Company Incorporated, 2013. ISBN: 9780321399526.

[16] File:RGB Colorcube Corner White.png - Wikimedia Commons. [online]. [cit. 2014-04-22]. Dostupné z: http://commons.wikimedia.org/wiki/File:RGB_Colorcube_Corner_White.png.

[17] File:YUV UV plane.svg - Wikipedia, the free encyclopedia. [online]. [cit. 2014-04-22]. Dostupné z: http://en.wikipedia.org/wiki/File:YUV_UV_plane.svg.

[18] Pixel and Planar Image Formats. Intel Corporation. [online]. [cit. 2014-04-27]. Dostupné z: https://software.intel.com/en-us/node/441488.

[19] *ISO/IEC 14496-10:2004(E).:* ISO Copyright Office, 2004.

[20] *ISO/IEC 14496-14:2003/Cor.1:2006(E).:* International Organization for Standardization, 2006.

[21] *ISO/IEC 14496-12:2005(E).:* ISO Copyright Office, 2005.

[22] Camera | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/hardware/Camera.html.

[23] Camera.PreviewCallback | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/hardware/Camera.PreviewCallback.html.

[24] GLSurfaceView | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/opengl/GLSurfaceView.html.

[25] GLSurfaceView.Renderer | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/opengl/GLSurfaceView.Renderer.html.

[26] SurfaceTexture | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/graphics/SurfaceTexture.html.

[27] SurfaceTexture.OnFrameAvailableListener | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/graphics/SurfaceTexture.OnFrameAvailableListener.html.

[28] Buffer | Android Developers. [online]. [cit. 2014-04-26]. Dostupné z: http://developer.android.com/reference/java/nio/Buffer.html.

[29] glReadPixles. Khronos Group. [online]. [cit. 2014-04-26]. Dostupné z: http://www.khronos.org/opengles/sdk/docs/man/xhtml/glReadPixels.xml.

[30] GLES20 | Android Developers. [online]. [cit. 2014-04-26]. Dostupné z: http://developer.android.com/reference/android/opengl/GLES20.html.

[31] MediaRecorder | Android Developers. [online]. [cit. 2013-10-11]. Dostupné z: http://developer.android.com/reference/android/media/MediaRecorder.html.

[32] CamcorderProfile | Android Developers. [online]. [cit. 2014-04-25]. Dostupné z: http://developer.android.com/reference/android/media/CamcorderProfile.html.

[33] FFmpeg. [online]. [cit. 2014-04-24]. Dostupné z: http://www.ffmpeg.org/.

[34] GNU Lesser General Public License v2.1 - GNU Project - Free Software Foundation. [online]. [cit. 2014-04-25]. Dostupné z: http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html.

[35] ffmpeg-1.0-android-arm.zip - javacv - FFmpeg 1.0 compiled for Android 2.2 - Java interface to OpenCV and more - Google Project Hosting. [online]. [cit. 2014-04-26]. Dostupné z: https://code.google.com/p/javacv/downloads/detail?name=ffmpeg-1.0-android-arm.zip.

[36] FFmpegFrameRecorder.java - javacv - Java interface to OpenCV and more - Google Project Hosting. [online]. [cit. 2014-04-26]. Dostupné z: https://code.google.com/p/javacv/source/browse/src/main/java/com/googlecode/javacv/FFmpegFrameRecorder.java.

[37] opencv/modules/highgui/src/cap_ffmpeg_impl.hpp at master · Itseez/opencv. [online]. [cit. 2014-04-26]. Dostupné z: https://github.com/Itseez/opencv/blob/master/modules/highgui/src/cap_ffmpeg_impl.hpp.

[38] FFmpeg: ffmpeg.c Source File. [online]. [cit. 2014-04-24]. Dostupné z: http://www.ffmpeg.org/doxygen/1.0/ffmpeg_8c-source.html.

[39] MediaCodec | Android Developers. [online]. [cit. 2014-04-24]. Dostupné z: http://developer.android.com/reference/android/media/MediaCodec.html.

[40] Surface | Android Developers. [online]. [cit. 2014-04-27]. Dostupné z: http://developer.android.com/reference/android/view/Surface.html.

[41] MediaCodecInfo.CodecCapabilities | Android Developers. [online]. [cit. 2014-047]. Dostupné z: http://developer.android.com/reference/android/media/MediaCodecInfo.CodecCapabilities.html.

[42] MediaCodec.BufferInfo | Android Developers. [online]. [cit. 2014-04-25]. Dostupné z: http://developer.android.com/reference/android/media/MediaCodec.BufferInfo.html.

[43] LinkedBlockingQueue | Android Developers. [online]. [cit. 2014-04-27]. Dostupné z: http://developer.android.com/reference/java/util/concurrent/LinkedBlockingQueue.html.

[44] Android MediaCodec stuff. [online]. [cit. 2014-04-27]. Dostupné z: http://bigflake.com/mediacodec/.

[45] grafika/src/com/android/grafika/RecordFBOActivity.java at master · google/grafika. [online]. [cit. 2014-04-28]. Dostupné z: https://github.com/google/grafika/blob/master/src/com/android/grafika/RecordFBOActivity.java.

[46] EGL - Native Platform Interface. Khronos Group. [online]. [cit. 2014-04-27]. Dostupné z: https://www.khronos.org/egl.

[47] Bitmap | Android Developers. [online]. [cit. 2014-04-30]. Dostupné z: http://developer.android.com/reference/android/graphics/Bitmap.html.

[48] Canvas | Android Developers. [online]. [cit. 2014-04-30]. Dostupné z: http://developer.android.com/reference/android/graphics/Canvas.html.

[49] GLUtils | Android Developers. [online]. [cit. 2014-04-30]. Dostupné z: http://developer.android.com/reference/android/opengl/GLUtils.html.

[50] X. Huang, A. Acero a H.W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, 2001. ISBN: 9780130226167.

[51] Basic concepts of speech - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 014-04-28]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts.

[52] S. Davis a P. Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 28.4 (srp. 1980), s. 357–366. ISSN: 0096-3518. DOI: 10.1109/TASSP.1980.1163420.

[53] CMU Sphinx - Speech Recognition Toolkit. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2013-10-05]. Dostupné z: http://cmusphinx.sourceforge.net/.

[54] Open-Source Large Vocabulary CSR Engine Julius. [online]. [cit. 2013-10-05]. Dostupné z: http://julius.sourceforge.jp/en_index.php.

[55] OpenEars - iPhone Voice Recognition and Text-To-Speech. [online]. [cit. 2013-10 -05]. Dostupné z: http://www.politepix.com/openears/.

[56] 'KALDI': About the Kaldi project. [online]. [cit. 2013-10-05]. Dostupné z: http://kaldi.sourceforge.net/.

[57] SpeechRecognizer | Android Developers. [online]. [cit. 2013-10-05]. Dostupné z: http://developer.android.com/reference/android/speech/SpeechRecognizer.html.

[58] Apache License, Version 2.0. [online]. [cit. 2014-04-29]. Dostupné z: http://www.apache.org/licenses/LICENSE-2.0.html.

[59] The BSD 2-Clause License | Open Source Initiative. [online]. [cit. 2014-04-29]. Dostupné z: http://opensource.org/licenses/bsd-license.php.

[60] Overview of CMUSphinx toolkit - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2013-10-05]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutorialoverview.

[61] AudioRecord | Android Developers. [online]. [cit. 2013-10-11]. Dostupné z: http://developer.android.com/reference/android/media/AudioRecord.html.

[62] Building application with pocketsphinx - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2013-10-05]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutorialpocketsphinx.

[63] MediaRecorder.AudioSource | Android Developers. [online]. [cit. 2013-10-11]. Dostupné z: http://developer.android.com/reference/android/media/MediaRecorder.AudioSource.html.

[64] Archived DR-BT21G : Bluetooth Headphones : Headphones : Sony Asia Pacific. [online]. [cit. 2014-01-10]. Dostupné z: http://www.sony-asia.com/product/dr-bt21g.

[65] AudioManager | Android Developers. [online]. [cit. 2013-10-11]. Dostupné z: http://developer.android.com/reference/android/media/AudioManager.html.

[66] TextToSpeech | Android Developers. [online]. [cit. 2014-05-07]. Dostupné z: http://developer.android.com/reference/android/speech/tts/TextToSpeech.html.

[67] Building Pocketsphinx On Android | CMU Sphinx - Speech Recognition Toolkit. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2013-10-05]. Dostupné z: http://cmusphinx.sourceforge.net/2011/05/building-pocketsphinx-on-android/.

[68] pocketsphinx on Android - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2014-02-02]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutorialandroid.

[69] Building Language Model - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2014-02-04]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutoriallm.

[70] Training Acoustic Model For CMUSphinx - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2014-02-04]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutorialam.

[71] pocketsphinxhandhelds - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2014-02-02]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/pocketsphinxhandhelds.

[72] Adapting the default acoustic model - CMUSphinx Wiki. CARNEGIE MELLON UNIVERSITY. [online]. [cit. 2013-10-05]. Dostupné z: http://cmusphinx.sourceforge.net/wiki/tutorialadapt.

[73] Microsoft WAVE soundfile format. [online]. [cit. 2013-11-20]. Dostupné z: https://ccrma.stanford.edu/courses/422/projects/WaveFormat/.

[74] Using Immersive Full-Screen Mode | Android Developers. [online]. [cit. 2014-05-01]. Dostupné z: https://developer.android.com/training/system-ui/immersive.html.

[75] Monitoring the Battery Level and Charging State | Android Developers. [online]. [cit. 2014-01-12]. Dostupné z: http://developer.android.com/training/monitoring-device-state/battery-monitoring.html.

[76] WindowManager.LayoutParams | Android Developers. [online]. [cit. 2014-04-21]. Dostupné z: http://developer.android.com/reference/android/view/WindowManager.LayoutParams.html#screenBrightness.

[77] Settings.System | Android Developers. [online]. [cit. 2014-04-21]. Dostupné z: http://developer.android.com/reference/android/provider/Settings.System.html.

[78] Avidemux - Main Page. [online]. [cit. 2014-04-30]. Dostupné z: http://fixounet.free.fr/avidemux/.