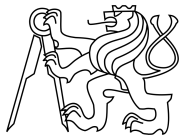
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Zobrazování filmových titulků na základě otisků zvukové stopy
Student:	Bc. Jakub Javůrek
Vedoucí:	doc. Ing. Ivan Šimeček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

1) Nastudujte problematiku časové synchronizace pomocí současných titulkových formátů (např. SRT, SSA...) s videem.
2) Popište problémy soudobých řešení (např. v důsledku prohozených scén či rozdílné snímkové frekvence).
3) Analyzujte existující možnosti jejich synchronizace.
4) Navrhněte titulkový formát, pro který je časování titulků udáváno primárně pomocí otisků získaných ze zvukové stopy pomocí technologie Waveprint [1].
5) Vlnkovou transformaci implementujte za použití CUDA (např. viz [2]).
6) S využitím knihovny libav [3] vytvořte jednoduchý software pro přehrávání videa s titulky používající tento formát.
7) Analyzujte možnost použití takového formátu pro promítání titulků souběžně s filmovou projekcí (např. pro neslyšící) a navrhněte řešení možných problémů.

Seznam odborné literatury

[1] http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=4217383
[2] http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=7394516
[3] https://libav.org/

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 5. května 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Zobrazování filmových titulků na základě otisků zvukové stopy

Bc. Jakub Javůrek

Katedra počítačových systémů
Vedoucí práce: doc. Ing. Ivan Šimeček, Ph.D.

8. ledna 2018

Poděkování

Předně děkuji doc. Ing. Ivanu Šimečkovi, Ph.D. za zpětnou vazbu a vedení této práce. Dále chci poděkovat své přítelkyní Pavle Taněvové a své rodině za podporu při studiu. Pavle Taněvové dále děkuji za pomoc při odstraňování gramatických chyb z textu této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. ledna 2018 .....

Odkaz na tuto práci

Javůrek, Jakub. *Zobrazování filmových titulků na základě otisků zvukové stopy.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce se zabývá časováním filmových titulků pomocí otisků získaných ze zvukové stopy. Součástí je analýza existujících titulkových formátů a problémů s jejich synchronizací s videem, popis existujících řešení těchto problémů a návrh formátu, ve kterém jsou titulky časovány primárně pomocí zvukových otisků. K tvorbě a porovnávání zvukových otisků je použita technologie Waveprint. Práce se dále zabývá akcelerací tvorby otisků na GPU pomocí CUDA. Výstupem je jednoduchý video přehrávač, který umožňuje přehrávat video společně s navrženým titulkovým formátem. V závěru se práce věnuje možnostem použití tohoto formátu pro promítání titulků souběžně s filmovou projekcí.

Klíčová slova titulky, tvorba zvukových otisků, Waveprint, CUDA, synchronizace titulků

Abstract

This thesis addresses movie subtitles matching using audio fingerprints. It consists of analysis of existing subtitle formats and problems that can be encountered when synchronizing them with video and description of existing solutions to these problems. The thesis proposes a subtitle format which, instead of timecode, uses audio fingerprints primarily. To create and compare

these fingerprints, Waveprint technology is used. This thesis also deals with accelerating the fingerprint creation on the GPU using CUDA. Source codes for a simple video player, capable of displaying subtitles in the proposed subtitle format are included. The thesis also contains analysis how to use this format when displaying subtitles with cinema movie projection simultaneously.

Keywords subtitles, audio fingerprinting, Waveprint, CUDA, subtitle synchronization

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Titulkové soubory umožňují neslyšícím a lidem neovládajícím daný cizí jazyk sledovat videa vytvořená kdekoliv na světě. Nestačí však jen to, že jsou ve správném jazyce. Většina současných formátů počítá se zobrazením jednotlivých titulků v pevně daném čase – timekódu. Problémem je, když je video v jiném formátu, než na který byly načasovány. Pak totiž může dojít k jejich desynchronizaci.

1.1 Možné problémy se synchronizací titulků

Problémy s časováním, které mohou při přehrávání titulků nastat, se dají shrnout do dvou kategorií:

1. Desynchronizace kvůli časování na jinou snímkovou frekvenci a

2. desynchronizace kvůli přidaným/odebraným scénám[1].

V případě, že jsou titulky přehrávány s videem s jinou snímkovou frekvencí[2], než na kterou byly časovány, dochází při přehrávání k jejich desynchronizaci, již musí řešit uživatel – buď stažením správné verze souboru (ať titulků, či videa), nebo pomocí nastavení přehrávacího programu, pokud to ten umožňuje.

Dalším možným problémem je, když některé scény v přehrávaném filmu chybí a jiné zase přebývají. Například při použití verze s přidanými scénami. Od nové scény jsou pak titulky stávající nepoužitelné. Toto je problém častý na filmových festivalech, kde jsou z externích zařízení přehrávány titulky pro neslyšící nebo cizince.

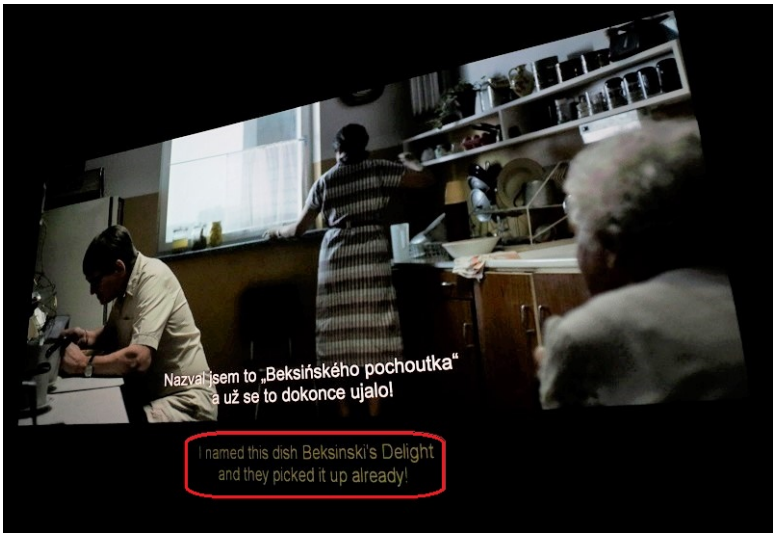
[1] Například režisérský sestřih filmu

[2] V celé práci je snímkovou frekvencí myšlena rychlost přehrávání videa – kolik snímků je za jednu vteřinu přehráno, kde celkový počet snímků ve videu je nezměněn.

1.2 Titulkování souběžně s filmovou projekcí

V předchozí sekci zmíněné problémy se týkají také filmových festivalů, na nichž jsou spolu s filmy promítány elektronické titulky [8] na k tomu určené plátno, umístěné pod tím filmovým. Tyto komplikace mohou být způsobeny například rozdílnou snímkovou frekvencí mezi digitální kopií poskytnutou překladateli filmu a filmem, který je dodán do kina. Stává se také, že tvůrce filmu těsně před jeho uvedením provede střihové změny – přidání, odebrání či prohození některých scén. To je problém při použití klasických formátů titulkových souborů, kde je každý titulek nasazen na statický čas.

V současnosti existují dvě metody, jak tyto problémy řešit, a obě vyžadují pracovníka obsluhy (dále titulkář), jenž musí být přítomen u každé projekce. Prvním způsobem je takzvané „klikání“ titulků, kde titulkář jednotlivé titulky potvrzuje k zobrazení manuálně [9]. V tomto případě nejsou soubory s titulky opatřeny timekódem.



Obrázek 1.1: Film se souběžně promítanými anglickými titulky na speciální plátno (zvýrazněno). [1]

Druhá metoda je inspirovaná klasickými titulkovými formáty. Využívá se zařízení, které čte titulkový formát opatřený timekódem. Titulkář zde musí zařízení ve správný moment spustit, a probíhá-li vše bez problému, titulky sedí přesně. Tento způsob však trpí stejnými slabinami jako při přehrávání videa na PC. Navíc, jak bylo řečeno dříve, neznáme předem povahu případných změn v kopii filmu, jež byla dodána k promítání. Dojde-li tedy k desynchronizaci během promítání, musí opět zasáhnout obsluha zařízení.

1.3 Cíl práce

Cílem práce je vytvořit titulkový formát, který jednotlivé titulky časuje pomocí otisků zvukové stopy. K tomu je nutné nejprve provést analýzu formátů existujících. součástí je také tvorba jednoduchého video přehrávače, který navržený titulkový formát dokáže použít k přehrávání titulků s videem.

Dalším krokem je nastudování a implementace techniky Waveprint, která byla vyvinuta pro identifikaci audia[3].

Analyzuji také možnosti implementace techniky Waveprint pomocí CUDA. Jedná se především o Haarovu vlnkovou transformaci, která je součástí techniky.

Na závěr analyzuji možnost použití vytvořeného formátu při přehrávání titulků z externího zařízení současně s filmovou projekcí.

1.4 Struktura práce

V kapitole 2 lze nalézt analýzu běžných titulkových formátů, včetně jejich problémů při přehrávání. Také analyzuji existující řešení těchto problémů – viz sekce 2.2.

Přehled použitých technologií se nachází v kapitole 3. Obsahuje popis použitých metod pro transformaci signálu, použitých knihoven a také technologie CUDA.

Návrh titulkového formátu se nachází v kapitole 4. Kapitola obsahuje také bližší popis toho, jak funguje technika Waveprint a jakým způsobem jsou použity technologie z kapitoly 3 při implementaci řešení.

V kapitole 5 se nachází zajímavé části implementace a jejich popis.

Kapitola 6 obsahuje výsledky měření.

V kapitole 7 lze nalézt analýzu možností použití titulkového formátu souběžně s filmovou projekcí a závěrečné shrnutí výsledků.

[3] Viz strana 15

Kapitola 2

Přehled současných řešení

2.1 Běžné titulkové formáty

V běžně používaných formátech jsou titulky časovány nějakou formou timekódu. Jako nejmenší jednotka jsou pak používány buď milisekundy, nebo snímky. Vzhledem k vzájemné podobnosti titulkových formátů jsem vybral tři z nich – SubRip, Society of Motion Picture and Television Engineering – Timed Text (dále SMPTE-TT) a MicroDVD. První používá jako nejmenší jednotku milisekundy, druhý snímky a třetí je na snímky přímo časován.

V následujícím textu tyto tři formáty krátce popíši a poté ukáži, že problémy, které se u nich mohou při synchronizaci vyskytnout, jsou jim společné.

2.1.1 SubRip

Formát SubRip je „asi nejznámějším titulkovacím formátem. Umí s ním pracovat snad všechny titulkovací programy“ [2]. Jedná se o textový formát. Co se časování týče, jsou časy zobrazení a skrytí uloženy ve formátu *hh:mm:ss,ms* (viz 2.1) – nejmenší hodnotou jsou milisekundy.

256
00:33:38,094 --> 00:33:42,485
3905 korun.
Zaplatíte hned?

257
00:33:43,094 --> 00:33:47,451
Ne, musím ještě do banky.
Hned jsem zpátky.

Zdroj: *Čestné modré oči, 5. epizoda*

Obrázek 2.1: SubRip formát [2]

Titulky v tomto formátu mohou být časovány s přesností na jednu milisekundu. Přesnost je ale zachována jen v případě, že jsou titulky nasazeny na přehrávanou verzi videa. Při přehrávání souboru jen s jinou snímkovou frekvencí se titulky časem desynchronizují.

2.1.2 SMPTE-TT

Jedná se o titulkový formát založený na XML, který je často používán v televizním vysílání [10]. Časy zobrazení a skrytí jednotlivých titulků jsou uloženy ve formátu *hh:mm:ss:frame* u návěstí *begin*, respektive *end* (viz 2.2). Nejmenší jednotkou je tedy jeden snímek, snímková frekvence je nastavena v hlavičce souboru [10].

```
  <body>
        <div>
            <p begin="01:00:06:12" end="01:00:08:15"
region="pop1" style="basic" tts:origin="17.5% 79.33%"
tts:extent="62.5% 5.33%">BARRY LINK: The next time</p>
            <p begin="01:00:06:12" end="01:00:08:15"
region="pop2" style="basic" tts:origin="20% 84.67%"
tts:extent="60% 5.33%">you're watching a video,</p>
            <p begin="01:00:08:15" end="01:00:10:28"
region="pop1" style="basic" tts:origin="25% 84.67%"
tts:extent="50% 5.33%">take the audio down.</p>
            <p begin="01:00:10:28" end="01:00:14:03"
region="pop1" style="basic" tts:origin="10% 84.67%"
tts:extent="80% 5.33%">Imagine you're hearing
impaired.</p>
```

Obrázek 2.2: SMPTE-TT formát [3]

Přestože se jedná o profesionální formát, trpí SMPTE-TT stejnými problémy jako formát SubRip. Správné časování titulků je závislé na použití správné verze videa. V profesionálním prostředí je totiž formát používán především kvůli tomu, že splňuje regulace týkající se datových toků, formátů atd. [11] [12]. Tím se však tako práce nezabývá.

2.1.3 MicroDVD

Textový formát podobný formátu SubRip s tím rozdílem, že časy zobrazení a skrytí titulků jsou reprezentovány číslem snímků (viz 2.3) [4]. Formát je tedy, na rozdíl od předchozích dvou formátů, nezávislý na změnách ve snímkové frekvenci mezi jinými verzemi videa.

V případě, že přehráváme verzi filmu s rozdílným celkovým počtem snímků, však dochází k desynchronizaci titulků – titulky jsou vázány na index snímku. Je pak jedno, zda je důvodem jeho změny přidání či odebrání scén z filmu,

```
{1}{93}Fantomas se zasmál|nejapnému a šovinistickému vtipu.
{94}{193}Druhý titulek
{194}{277}Třetí titulek
```

Obrázek 2.3: MicroDVD formát [4]

nebo duplikace či odebrání některých snímků a následné zvýšení, respektive snížení rychlosti přehrávání. V dnešní době se již moc nevyužívá [4].

2.2 Existující řešení problému se synchronizací

Běžné titulkové formáty synchronizaci titulků s videem neřeší – počítají s nezměněným formátem videa.

Způsobem, jak takové titulky synchronizovat automaticky, se zabývá práce [5]. Problém je zde řešen vkládáním zvukových signatur alespoň na dvě místa v souboru. Výsledný soubor v podstatě dodržuje formát SubRip, pouze je obohacen o zvukové otisky (viz 2.4).

Metoda autorů spočívá v tom, že před spuštěním přehrávání je zvuková stopa videa zpracována a z časů, které jsou uvedeny u zvukových signatur v titulkovém souboru, jsou signatury vypočítány také. Provede se srovnání, a pokud se páry signatur neshodují, časy titulků jsou přepočítány až do výskytu první signatury. V případě přetrvávající nerovnosti signatur se přepočítají titulky v časech mezi první a druhou signaturou. Při dalších časových nesrovnalostech se postupuje analogicky s pomocí dalších signatur.

```
01. 0
02. 00:00:00,000 --> 00:00:00,000
03. @fingerprint@ -177,-168,-167,...,934,934,904
04.
05. 1
06. 00:01:46,144 --> 00:01:48,831
07. - ( woman laughing )
```

Obrázek 2.4: SubRip formát obohacený o zvukové otisky (zvýrazněno) [5]

Takto je zautomatizováno řešení problému synchronizace s videem s jinou frekvencí zobrazování snímků. Může se však také stát, že se verze souboru, na niž byly titulky původně nasazeny, liší od té přehrávané obsahem – například režisérský sestřih filmu. V případě prohozených scén bude metoda autorů úspěšná, obsahuje-li soubor s titulky dostatek zvukových otisků na správných místech.

Ovšem může se stát, že některé scény v přehrávaném filmu chybí a jiné zase přebývají. V případě přebývajících scén by metoda autorů byla úspěšná, pokud

by zvukové otisky byly umístěny ve správných časech, tj. okolo přidaných scén tak, aby byly dostupné titulky posunuty na správný čas. Nové scény pak zůstanou bez titulků. Toto však vyžaduje předchozí znalost umístění a délky nového obsahu. Pokud by naopak některé scény chyběly, nebude vzhledem k přebývajícím titulkům tato metoda fungovat správně.

Tento problém vzniká tím, že se jako primární identifikátor pozice titulku ve videu je používán timekód – zvukové otisky slouží pouze k synchronizaci, jejíž kvalita je závislá na jejich počtu a umístění. Za normálních podmínek bude tato metoda fungovat dobře, avšak při přestříhaném filmu naráží na limity dané sekvenčností klasických titulkových formátů – titulky jsou řazeny dle času. V případě filmových festivalů či speciálních projekcí[4] pak k těmto problémům dochází ve větší míře než v domácím prostředí.

Pokud tedy nemáme předchozí znalost nové verze filmu, neřeší ani tato metoda problém při výměně scén.

2.3 Zobrazování titulků na základě zvukové stopy

Metoda z článku [5] tedy sice do jisté míry řeší desynchronizaci při jiné snímkové frekvenci, ale při změně scén ve filmu naráží na stejné problémy, jako běžné titulkové formáty.

Problémem je totiž závislost titulků na čase – jako řešení tedy navrhuji titulkový formát, kde je timekód nahrazen zvukovými otisky. Vhodný titulek je vybírán podle shody se zvukovým otiskem s minimální závislostí na čase.

Pro tvorbu otisků v titulkovém souboru je použita technika založená na dekompozici na vlnky a následné uložení jejich signatur v podobě minhashů tak, jak je popsáno v [7]

[4] Například pro neslyšící [13]

Kapitola 3

Přehled použitých technologií

Jak bylo uvedeno v kapitole 1 (viz strana 1), je cílem této práce vytvořit titulkový formát časovaný pomocí otisků zvukové stopy. V této kapitole jsou uvedeny a krátce popsány technologie, které k tomuto účelu využívám.

3.1 Metody transformace signálu

Již ze zadání práce vyplývá nutnost použít při jeho plnění algoritmy pro analýzu zvukového signálu. Jsou jimi Fourierova transformace a Haarova vlnková transformace. Obě jsou pak použity v implementaci techniky Waveprint, které je věnována sekce 4.2.

3.1.1 Haarova vlnková transformace

Vlnkovou transformací, kterou používám v implementaci, je Haarova vlnková transformace. Jedná se o nejjednodušší a nejstarší transformaci tohoto typu. Informaci na vstupu rozkládá sub-signály o poloviční délce [14].

Vlnková transformace umožňuje zachytit okamžité, lokální změny v signálu. Vlnka je v podstatě část signálu, izolovaná od zbytku v určitém čase a frekvenci. Jedním z jejích hlavních použití je komprese signálu [15].

Pomocí vlnkové transformace lze ze signálu získat informace o tom, které frekvence byly v který čas nejvýznačnější. Oproti tomu lze použitím Fourierovy transformace získat jednotlivé frekvence, ze kterých je signál složen, ale bez časové lokalizace. Vlnková transformace tedy umožňuje zachovat nejvýraznější části signálu společně s jejich změnami v čase – signál komprimuje.

Výpočet jednorozměrné transformace je proveden následovně [15]:

1. Vstupní vektor uvažujeme jako vektor párů (a, b), kde a je hodnota na sudých pozicích, b na lichých. Velikost vektoru předpokládáme rovnou mocnině dvou

2. První polovinu vektoru nahradíme hodnotami získanými výpočtem: $\frac{(a+b)}{\sqrt{2}}$

3. Druhou polovinu vektoru nahradíme hodnotami získanými výpočtem: $\frac{(a-b)}{\sqrt{2}}$

4. Postup opakujeme na první polovině vektoru, dokud tato nemá velikost rovnu jedné

Celý výpočet lze ilustrovat na následujícím příkladu, se vstupními hodnotami uvedenými v tabulce 3.1:

Tabulka 3.1: Vektor vstupních hodnot

1.0	2.0	3.0	1.0	2.0	6.0	1.0	1.5

Vstupní hodnoty rozdělíme na dvojice dle sudých a lichých pozic:

Tabulka 3.2: Rozdělení vstupních hodnot

(1.0, 2.0)	(3.0, 1.0)	(2.0, 6.0)	(1.0, 1.5)

Vezmu první dvojici, její hodnoty sečtu, vydělím $\sqrt{2}$ a uložím do původního vstupního vektoru na příslušnou pozici, v tomto případě první. Poté hodnoty odečtu a také vydělím $\sqrt{2}$. Výsledek uložím na první pozici v druhé polovině vstupního vektoru. Postup opakuji pro každou dvojici.

Výsledkem je nový vektor. Hodnoty získané výpočtem $\frac{(a+b)}{\sqrt{2}}$ jsou umístěny v jeho první polovině (modře), hodnoty získané z $\frac{(a-b)}{\sqrt{2}}$ pak v druhé polovině (červeně):

Tabulka 3.3: Výsledek první iterace algoritmu

2.12	2.83	5.66	1.77	-0.71	1.41	-2.83	-0.35

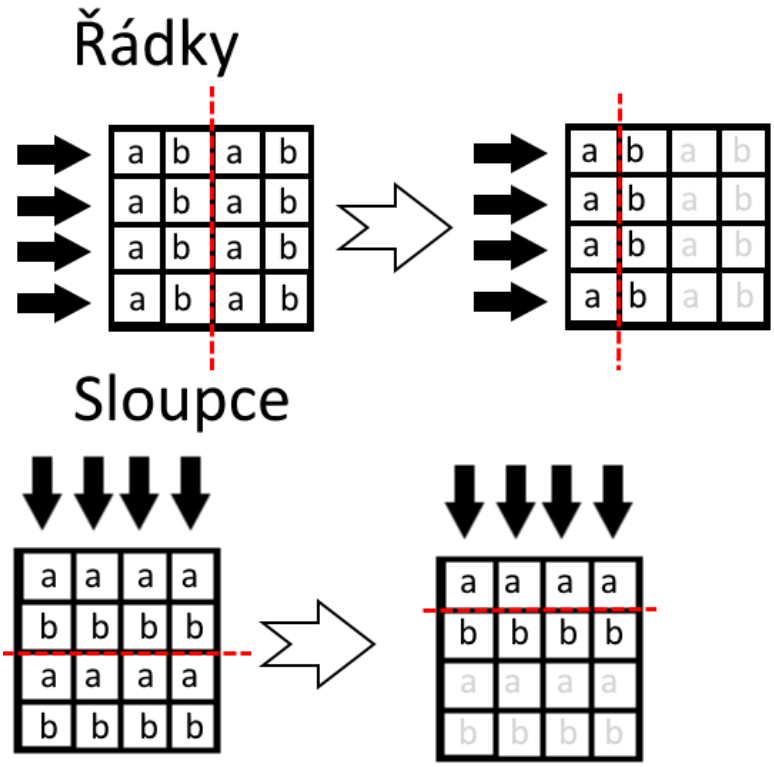
Ze získaného vektoru následně vezmu hodnoty v první polovině. Tyto hodnoty jsou novým vstupním vektorem, nad nímž postup opakuji, dokud mi po rozpůlení výsledného vektoru nezbyde jediné číslo. Výsledkem je pak v tomto případě následující vektor:

Tabulka 3.4: Výsledek Haarovy vlnkové transformace

6.19	-1.24	-0.5	2.76	-0.71	1.41	-2.83	-0.35

V případě, že potřebuji transformaci provést nad dvourozměrnými daty, provádím dvourozměrnou vlnkovou transformaci. Té je dosaženo nejprve provedením jednorozměrné transformace na každý jednotlivý řádek matice vstupních dat. Poté je ve výsledné matici stejným způsobem transformován také

každý její sloupec. Tento postup ilustruje obrázek 3.1 – hodnoty jsou rozděleny do dvojic, poté jsou transformovány řádky, dále pak sloupce matice. Výsledky jsou uloženy do příslušných polovin matice (zvýrazněno červenou čarou).



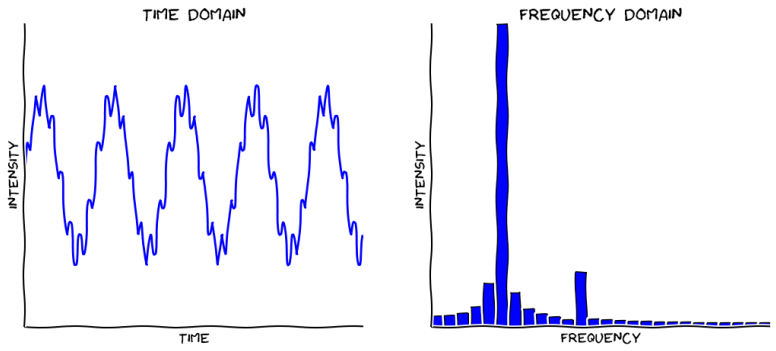
Obrázek 3.1: Dvourozměrná vlnková transformace.

3.1.2 Fourierova transformace

Metoda, jak signál transformovat z časové reprezentace do reprezentace frekvenční – signál je rozložen do svých jednotlivých frekvenčních složek. Výsledkem je vektor komplexních čísel, obsahující intenzity[5] jednotlivých frekvencí v reálné složce. Imaginární složky pak obsahují fázi. Frekvence jsou udány indexem, na kterém se dané komplexní číslo nachází.

Vstupní signál si lze představit jako složení vln tvořených funkcemi sinus a kosinus o různých frekvencích, amplitudách a fázích. Tento signál lze pak chápat jako funkci, která vrací velikost amplitudy v určitém čase. Pomocí Fourierovy transformace lze zjistit, jakou má určitá frekvence ve zdrojovém signálu intenzitu.

[5]Nebo také amplitudy



Obrázek 3.2: Fourierova transformace [6]

Použitím Fourierovy transformace sice ze signálu získáme jeho jednotlivé frekvence a jejich amplitudy, nelze je však lokalizovat v čase. Tento problém lze do jisté míry eliminovat jejím použitím na vícero úseků vstupního signálu, které se v ideálním případě částečně překrývají. Tak lze vytvořit spektrogram v určitém frekvenčním rozsahu, který popisuje změny zvukových frekvencí v závislosti na čase [16]. Délka jednotlivých úseků analyzovaného signálu pak určuje přesnost, s jakou lze k jednotlivým časům přiřadit frekvenční hodnoty – čím kratší časové úseky analyzujeme, tím vyšší rozlišení získáme.

3.2 Knihovny pro práci s audiem a videem

Pro načtení audio a video paketů ze souboru a jejich konverzi do potřebných formátů používám knihovnu LibAV. Jedná se o open-source multiplatformní knihovnu, která umožňuje načíst, zpracovat, konvertovat a jinak manipulovat mediální datové streamy. Podporuje velké množství formátů a protokolů [17].

Tato knihovna však sama neumožňuje přehrání zvuku či videa na výstupu. K tomu účelu používám knihovnu SDL 2. Tato knihovnu poskytuje nízko úrovňový přístup ke zvukové kartě, myši, klávesnici a dalším. Umožňuje pracovat s grafickým hardware skrz OpenGL a Direct3D. Znovu se jedná o open-source multiplatformní knihovnu [18]. Také obsahuje funkce pro tvorbu a správu vláken.

3.3 CUDA

Výpočty prováděné algoritmem Waveprint jsou časové náročné, proto tento algoritmus implementuji v CUDA. Jedná se o programovací model vyvinutý společností NVIDIA, umožňující psát v jazycích C, C++, a dalších[6] kód prová-

[6]Fortran, Python, MATLAB

děný na grafickém procesoru karet od společnosti NVIDIA. Zatímco sekvenční části programu běží na CPU, mohou pomocí CUDA probíhat náročné výpočty paralelně na GPU [19].

Kód, který je spuštěn na GPU, se nazývá kernel. Kernel je prováděn paralelně vícero vlákny, často se jedná o tisíce. Vlákna jsou organizována do bloků, uvnitř kterého je možné je synchronizovat a mohou sdílet data. Počet vláken uvnitř bloku je omezen verzí výpočetních možností grafické karty[7] [20]. Bloky jsou pak organizovány do mřížky.

Paměť hostujícího stroje a GPU není sdílená, proto je nutné vstupní data do paměti GPU nejprve přenést. Stejně tak je nutné výsledky přenést zpět do paměti hostujícího stroje. Tyto přenosy mohou tvořit úzké hrdlo při provádění programu, je tedy dobré je buď omezit, nebo, pokud možno, paralelizovat pomocí tzv. streamů. Každý stream provádí sobě přiřazené operace sekvenčně v pořadí, v jakém byly zadány. Lze ale alokovat vícero streamů, které pak jim přiřazené operace provádí nezávisle na sobě [20].

CUDA SDK je dodáváno v různými knihovnami, které implementují často používané operace. V práci používám dvě z těchto knihoven:

- Thrust – knihovna implementující řadu základních operací s daty v paralelním prostředí (řazení, prefixový součet, redukce,...) [21]
- cuFFT – knihovna implementující rychlou Fourierovu transformaci (FFT) akcelerovanou na GPU [22]

[7] Od verze 3.0 je to 1024 vláken na jeden blok

Kapitola 4

Analýza a návrh

4.1 Návrh titulkového formátu

Při návrhu jsem, stejně jako autoři studie na [5], vycházel z formátu SubRip. V novém formátu jsou ale titulky časovány primárně podle zvukové složky videa – časy zobrazení a skrytí titulku jsou nahrazeny hashem zvuku, který titulku těsně předchází.

Časovou složku, kterou reprezentuji v milisekundách, jsem ve formátu nakonec ponechal. Hashe zvukové složky a časový kód je tak možné zatížit váhami, které určí procentuální důležitost každé ze složek při srovnávání otisků. Časové složce lze nastavit nulovou váhu. Toto je pro případy, kdy různým dialogům předchází podobné audio (např. hudba ve filmu) – aby nedošlo k falešným pozitivním identifikacím. Časy jsou porovnávány jednoduše rozdílem času právě zpracovávaného zvuku a času, který je uložen v souboru s titulky.

Titulkový formát má pak následující podobu:

```
1 Index titulku
Časy počátku a konce v milisekundách, oddělené znakem '_'
3 Celočíselná signatura, její hodnoty odděleny znakem ';'
Text prvního řádku
5 [Nepovinný text druhého řádku]
```

Kód 4.1: Titulkový formát fsrt

Jelikož vycházím z formátu SubRip (přípona .srt), nazývám tento nový formát Fingerprinted SRT (dále „fsrt“).

4.2 Waveprint

Algoritmus Waveprint [7] je stěžejní součástí implementace. Při hledání shod u zvukových dat totiž nelze spoléhat na jednoduché přístupy typu porovnávání spektrogramů, nebo dokonce přímé srovnávání zdrojových dat. Dvě zvukové

stopy, které jsou až na kvalitu nebo použité kodeky[8] stejné, se budou při použití takových metod jevit jako rozdílné [7]. Proto bude k tvorbě zvukových otisků a jejich porovnávání použita technika Waveprint.

Tato technika, založená na počítačové vizi, spočívá ve využití vlnek k rychlé extrakci obrazových dat z velkých databází [7] [23]. Ze zdrojového signálu jsou extrahovány vlnky a následně jsou vybrány ty nejvýznačnější – takové, které ho nejvíce popisují.

4.2.1 Tvorba otisků

Z vzorků extrahovaných z audio stopy je vytvořen spektrogram, reprezentující změny zvukových frekvencí v závislosti na čase. Ten je dle [7] rozdělen po 32 frekvenčních hodnotách, od 318 Hz do 2 kHz – tato frekvenční pásma dostatečně reprezentují obsah vstupního signálu pro účely algoritmu. Rozdělení je provedeno po 371 milisekund dlouhých úsecích s odstupy délky 11,6 milisekund. Spektrogram díky tomu lépe postihuje změny v čase [7].

Spektrogram je následně rozdělen do několika překrývajících se úseků, jejichž délka je dána nastavení, je však násobkem 11,6 milisekund. Mezi vybranými daty je při tvorbě databáze rozestup pevně daný, při tvorbě signatury, kterou pak hledám, jsou náhodné [7]. Na získaných datech je poté prováděna dvourozměrná vlnková transformace, která signál převádí do formy, ve které jsou jeho vlastnosti čitelnější – vlnka je vlastně signál v lokálním měřítku [15].

Z výsledků jsou vždy vybrány vlnky s nejvyšší absolutní hodnotou[9] [7], ostatní hodnoty jsou považovány za nulové. Stačí uložit pouze jejich znaménka, nebo jestli se jedná o nulovou hodnotu [23]. Počet vybraných vlnek je věcí nastavení, autoři metody doporučují 200 [7].

Výsledná data z jednotlivých úseků jsou pak hashována pomocí minhash algoritmu – techniky, s jejíž pomocí lze rychle určit míru podobnosti dvou množin [24]. Hash je v ní reprezentován množinou sub-hashů. Zde stačí v různých permutacích vektoru s vlnkami nalézt index první (a tedy minimální) nenulové hodnoty. Dle [25] předpokládáme, že nejvyšší hodnota, která se v minhashi může objevit, je číslo 255 – na jeho reprezentaci stačí jeden bajt.

Každý minhash je nakonec rozložen do l skupin o c hodnotách (jednotlivá čísla mají velikost jednoho bajtu) a jejich bitové reprezentace zřetězíme. Výsledkem je číslo reprezentované c bajty [25].

Výsledkem je signatura tvořená celými čísly, která zdrojová data popisuje. Počet celočíselných hodnot, které signaturu tvoří, je dán nastavením algoritmu – velikostí[10] jednotlivých hodnot a počtem vzorků, které byly k tvorbě otisku použity.

[8] Software sloužící ke **ko**mpresi či **dek**ompresi mediálního obsahu. Pro video například h.264, DivX aj.

[9] Tedy ty s nejvyšší amplitudou – nejvýznačnější

[10] Počtem bajtů, použitých k jeho reprezentaci

4.2.2 Porovnávání otisků

Předtím, než je možné otisky porovnávat, je nutné vytvořit databázi otisků. Tu vytvořím následujícím způsobem:

1. Načtu titulky s jejich signaturami ze souboru
2. Každou ze signatur rozdělím do l množin s indexy *1 až* l
3. Množiny se stejným indexem sjednotím, vznikne tak l nových množin
4. Pro každé celé číslo z každé z těchto l množin:
    a) Přiřadím k číslu každý titulek, v jehož původní signatuře se toto číslo vyskytuje

Výsledkem je l tabulek indexovaných od *1* do l, s řádky ve tvaru: *celé číslo: titulek*.

Za předpokladu, že máme vytvořenou databázi otisků, ve které chceme nalézt shodu s otiskem nějakého zvukového signálu, lze jejich srovnání dle [26] provést v následujících krocích:

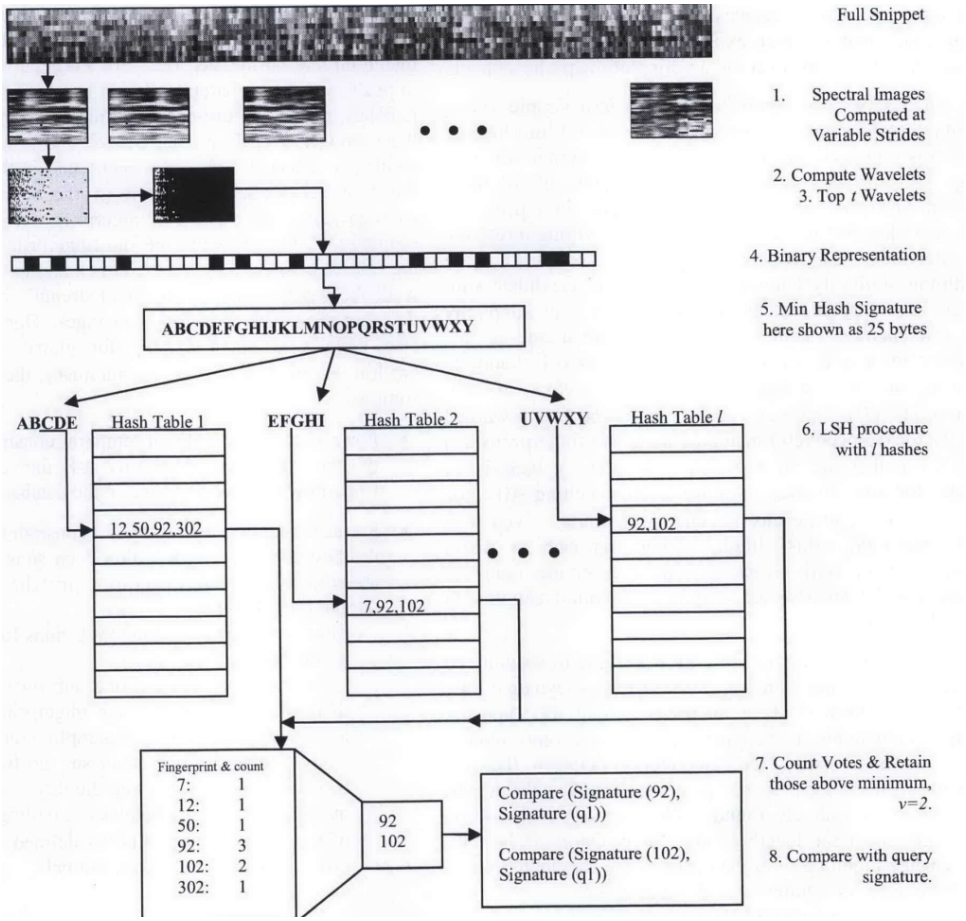
1. Ze vstupních dat vytvořím dle sekce 4.2.1 spektrogram
2. Spektrogram rozdělím do částí dělitelných 11,6 ms
3. Pro každou z částí pak:
    a) Provedu vlnkovou transformaci
    b) Označím t význačných vlnek (těch s nejvyššími absolutními hodnotami)
    c) Pomocí minhash algoritmu vytvořím jednotlivé sub-hashe z vlnek (první výskyt význačné vlnky v různých permutacích)
    d) Jednotlivé sub-hashe rozdělím do l množin o c hodnotách
    e) Bitové reprezentace všech čísel uvnitř jednotlivých množin sloučím. výsledkem je l čísel indexovaných od *1* do l
    f) Pro každý titulek pak:
        i. Za každé číslo, které vyskytuje v tabulce se stejným indexem, počítám jeden hlas
        ii. Pokud je počet hlasů vyšší než předem zvolená mez h[11], označím titulek jako kandidáta na pozitivní nález
        iii. Spočítám Hammingovu vzdálenost mezi relevantní částí signatury kandidátního titulku a všemi čísly, která se v tabulkách vyskytovala

[11] Hodnota meze je mezi *1* a l

4. Pro každý z titulků sečtu jeho Hammingovy vzdálenosti. Výsledky vydělím délkou signatury a vyberu tu nejvyšší hodnotu

Výsledkem je reálné číslo v intervalu $[0, 1]$, udávající jistotu v ekvivalenci dvou srovnávaných otisků.

Celý proces porovnání otisků je ilustrován na následujícím schématu z [7]:



Obrázek 4.1: Schéma procesu porovnání otisků v algoritmu Waveprint [7]

4.3 Přehrávač videa

Při tvorbě přehrávače videa jsem postupoval především podle návodu na [27] a manuálu pro knihovnu LibAV [17], kterou používám pro dekódování audio a video stop. K jejich zobrazení je pak použito knihovny SDL 2 [18].

Jelikož přehrávání zvuku a videa je rozděleno mezi zvukovou kartu a GPU, nejsou tyto dvě složky automaticky synchronizovány. K jejich synchronizaci lze využít informace obsažené v paketech video stopy. Každý paket obsahuje hodnoty *Presentation Time Stamp (pts)* [28]. Jedná se o čas, ve kterém má být snímek vykreslen – kvůli kompresi nemusí být snímky v souboru vždy uloženy v pořadí, v jakém mají být přehrávány. Extrahované snímky tedy uložím a podle hodnoty *pts* zobrazuji v závislosti na čase přehrávání zvukové složky.

Jak již bylo řečeno, titulky ve formátu *fsrt* dekóduji pomocí algoritmu Waveprint. Jelikož algoritmus potřebuje na vstupu zvuková data ve specifickém formátu vzorkování, jsou tato data předem extrahována, převzorkována pomocí knihovny LibAVresample[12] a uložena v paměti. Na úkor paměti RAM tak šetřím procesorový čas. Zároveň se tak vyhýbám situaci, kdy dochází k nečinnosti GPU z důvodu nedostatku vstupních dat.

Samotný přehrávač pracuje se třemi vlákny (na CPU) – jedno pro dekodér audia a videa, druhé pro zobrazení výstupu na obrazovku a třetí pro dekódování titulků pomocí algoritmu Waveprint. Vlákna vytvářím a spravuji pomocí funkcí, obsažených v knihovně SDL 2.

4.4 Akcelerace na GPU

Snímková frekvence filmu se standardně pohybuje okolo 24 a 25 snímků za vteřinu [29]. Pro zachování přesnosti na jeden snímek je tedy nutné provést algoritmus Waveprint spolu s porovnáním výsledku s titulkem až 25krát za vteřinu. Vzhledem k výpočetní složitosti algoritmu není na CPU možné tuto přesnost zachovat při současném přehrávání videa. Proto jsem se rozhodl algoritmus implementovat na GPU pomocí CUDA.

4.4.1 Vlnková transformace pomocí CUDA

Vlnková transformace je důležitou součástí algoritmu Waveprint. Nejvýznačnější vlnky tvoří základ při tvorbě zvukových otisků – viz sekce 4.2.

Jelikož vstupní data jsou dvourozměrná, musím použít dvourozměrnou vlnkovou transformaci. Její implementaci lze rozdělit do dvou kernelů: jeden pro řádkovou a druhý pro sloupcovou transformaci. Na vstupní data pak nahlížíme jako na dvojice hodnot (a, b). Každé vlákno pak bude provádět obě výpočetní operace[13] nad jemu přiřazenou dvojicí (a, b).

[12] Součást knihovny LibAV

[13] Viz seznam na straně 10, položky 2 a 3

Vstupní data mají dvourozměrnou podobu, nabízí se tedy bloky vláken rozdělit do dvourozměrné mřížky, kde globální index vlákna na x-ové souřadnici určuje sloupec, na y-ové řádek matice vstupních dat.

Velikost vstupních dat může jednoduše přesáhnout počet vláken v jednom bloku. A synchronizace přes vícero bloků uvnitř kernelu není dost dobře možná. Proto je uvnitř kernelu proveden vždy jen jeden cyklus vlnkové transformace. Poté je ukončen, výsledky uloženy a kernel je znovu zavolán s novými parametry.

Aby nedošlo k write-after-read hazardu, je nutné výsledky ukládat do výstupního pole, z nějž jsou po ukončení kernelu relevantní výsledky překopírovány do vstupních dat.

Bližší popis lze nalézt v kapitole 5.

4.4.2 Waveprint v CUDA

Co se akcelerace na GPU týče, součástí zadání pro tuto práci byla pouze vlnková transformace. Jelikož ta je součástí algoritmu Waveprint, spatřuji výhody v převodu většiny tohoto algoritmu do CUDA. Jednou z nich je omezení přenosů mezi pamětí GPU a hostujícím strojem, ale jsou i další[14].

Prvně je nutné vytvořit spektrogram, k čemuž je nutné provést na vstupních datech Fourierovu transformaci [16]. Použiji již hotovou implementaci v knihovnách CUDA – cuFFT [22].

Výsledek transformace je pak ještě potřeba převést do logaritmického měřítka. Jelikož toho je dosaženo výpočtem průměrných hodnot v daných úsecích spektrogramu, lze převod provést pomocí kernelu na bázi paralelní redukce.

Pak jsou zavolány dříve zmíněné kernely vlnkové transformace. Po jejím provedení je z výsledků vybráno v nejvýznačnějších vlnek – takových, které mají nejvyšší absolutní hodnotu. K tomu použiji řadících funkcí z knihovny Thrust [21].

K samotnému výpočtu minhashové signatury lze použít paralelní redukce s hledáním minima.

Podrobnosti implementace lze nalézt v kapitole 5.

[14] viz Sekce 4.4

Kapitola 5

Realizace

V této kapitole jsou uvedeny a popsány důležité či zajímavé části mé implementace, především algoritmu Waveprint. Kompletní zdrojové kódy lze nalézt v příloze.

5.1 Parametry algoritmu Waveprint

Základní parametry algoritmu jsem ponechal tak, jak jsou doporučeny autory prací na [7] a [25]. To znamená, že pracuji se zvukovou stopou převzorkovanou do mono o frekvenci 2 kHz. Při extrakci význačných vlnek získávám prvních 200.

Stejně jako v práci na [25] jsou v mé implementaci minhashe tvořící jednotlivé sub-hashe složeny ze 100 čísel v rozmezí $[0, 255]$. Také jsou poté ve skupinách po 4 sloučeny jejich bitové reprezentace. Výsledky jsou pak rozděleny do 25 tabulek způsobem popsaným v sekci 4.2.2 (l je rovno 25).

Procentuální jistota v nalezený titulek, při které je tento titulek přijat, byla nastavena na 55 %, při hledání konce titulku na 40 %[15]. Zvýší se tak sice pravděpodobnost falešných pozitiv, zároveň se však zvýší pravděpodobnost pravých pozitiv v signálu se sníženou kvalitou – například s přidaným šumem. Ze stejných důvodů byl počet hlasů při hledání kandidátních titulků[16] nastaven na 2.

Autoři v práci na [25] doporučují použití alespoň 10 vteřin dlouhých úseků zvukových dat při porovnávání. Pro zvýšení efektivnosti i více (30 či 60 vteřin). Při použití algoritmu pro nasazení titulků paralelně s přehráváním videa pak bez častého bufferovaní video nakonec „dostihne“ titulky. Výsledek je ten, že titulky, které jsou od určitého bodu nasazeny, již měly být zobrazeny v dřívější čas, než ve kterém se nachází přehrávač. Titulky by pak nebyly zobrazeny.

[15] Hledám totiž konec pouze jednoho titulku – pravděpodobnost falešných pozitiv je velmi nízká

[16] Viz sekce 4.2.2

Zvolil jsem proto použití úseků dlouhých 5 vteřin. Zvýší se tak sice výskyt falešných pozitiv, video lze však přehrát s titulky bez problémů[17].

I když je při nasazování titulků použit algoritmus Waveprint, pokud je titulek s nalezeným začátkem (ale ne koncem) delší než určité k, je hledání jeho konce zastaveno a délka titulku je položena rovna k. Za k jsem zvolil hodnotu 8000 milisekund, jíž považuji za dostatečnou dobu pro přečtení většiny možných titulků.

5.2 Implementace algoritmu Waveprint

Jak bylo řečeno v kapitole 4, sekce 4.4.2, snažil jsem se implementovat co největší část algoritmu Waveprint pomocí CUDA. Tvorba otisků je až na poslední krok (rozdělení do skupin dle sekce 4.2.1) v CUDA implementována. Jejich porovnávání jsem z časových důvodů již na GPU nepřenesl.

Algoritmus Waveprint sestává z vícero částí, které jsou prováděny postupně: tvorba spektrogramu, dvourozměrná vlnková transformace, extrakce význačných vlnek, tvorba minhashových sub-signatur a případné porovnání dvou otisků.

Jednotlivé části algoritmu jsou popsány v pořadí jejich provedení.

5.2.1 Tvorba spektrogramu

Nejprve ze vstupního signálu vyberu 5 vteřin dlouhý úsek v okolí daného časového bodu[18] tak, že se zvolený časový bod nachází přesně uprostřed tohoto úseku. Je-li pak časovým bodem začátek či konec titulku, obsahuje tato část signálu jak mluvené slovo jemu odpovídající, tak část vzorků z okolí titulku. Snažím se tak získat mix vzorků, jenž bude pro každý titulek co nejvíce unikátní.

Ze získané části signálu pak generuji spektrogram. Ten vytvářím pomocí krátkodobé Fourierovy transformace [30]. Vstupní signál si dle [7] rozdělím do okének velikosti 371 ms, na kterých pak provádím rychlou Fourierovu transformaci pomocí knihovny cuFFT.

[17] Toto platí alespoň na stroji s parametry dle tabulky 6.1, nebo výkonnějším.

[18] Zde začátek či konec titulku

```
void Waveprint::createLogSpectrogram(double *samples, size_t
    samplesLen, double *d_spectrum)
{
  cudaMemsetAsync(d_spectrum, 0, spectrogramWidth * config.
    numOfLogBins * sizeof(double), memsetStream);
  cufftComplex *samplesInComplexFormat = new cufftComplex[
    frameSize * spectrogramWidth]; //data vzorku v komplexnich
    cislech

  for (int i = 0; i < spectrogramWidth; i++)
  {
    for (int j = 0; j < frameSize; j++)
    {
      samplesInComplexFormat[i * j + j].x = samples[i*step + j];
      samplesInComplexFormat[i * j + j].y = 0;
    }
  }

  cudaMemcpy(d_samplesInComplexFormat, samplesInComplexFormat,
    frameSize * spectrogramWidth * sizeof(cufftComplex),
    cudaMemcpyHostToDevice);

  if (cufftExecC2C(plan, d_samplesInComplexFormat,
    d_samplesInComplexFormat, CUFFT_FORWARD) != CUFFT_SUCCESS) {
    std::cerr << "Failed to execute cuFFT plan!" << std::endl;
    exit(1);
  }
  cudaDeviceSynchronize();
  delete[] samplesInComplexFormat;

  for (int i = 0; i < spectrogramWidth; i++)
  {
    convertSpectrumToLogScale_call(d_samplesInComplexFormat + (i *
     frameSize), frameSize,
      d_spectrum, config.numOfLogBins, i, cudaProperties.
    maxThreadsPerBlock, spectrogramCreationStreams[i]); //rozdel
    frekv. spektrum do 32 binu
  }
  cudaDeviceSynchronize();
}
```

Kód 5.1: Tvorba spektrogramu

Po dokončení transformace získaná data rozdělím dle [7] do 32 frekvenčních skupin od 318 Hz do 2 kHz. Implementace viz přiložený zdrojový kód.

Výsledkem je spektrogram, který dále rozdělím do několika stejně velkých, překrývajících se částí, nad nimiž pak provádím vlnkovou transformaci a tvořím z nich jednotlivé sub-hashe. Každá z částí má délku násobku 11,6 ms. Dobrou hodnotou se ukazuje být přibližně 64 ms.

5.2.2 Dvourozměrná vlnková transformace

Implementovat vlnkovou transformaci pomocí CUDA bylo součástí zadání práce.

Implementace sestává ze čtyř kernelů: transformace řádků, transformace sloupců a dvou kernelů, které kopírují potřebná data z výstupních polí na vstup pro další iteraci.

Následující zdrojový kód patří kernelu Haarovy vlnové transformace pro řádky vstupní matice. Kernel je prováděn vlákny rozdělenými do dvourozměrných bloků ve dvourozměrné mřížce – vlákna jsou adresována pomocí indexů X (sloupce) a Y (řádky). Počet řádků se v jednotlivých iteracích nemění. Transformace je prováděna pro všechny řádky současně.

Každé vlákno provádí výpočty nad dvěma vstupními hodnotami ze vstupních dat (viz kapitola 3, sekce 3.1.1). Pro první iteraci je tedy kernel volán s celkovým počtem vláken $\frac{početsloupcůmaticevstupníchdat}{2} * počet řádků$. Každá další iterace má počet vláken o polovinu nižší než ta předchozí. Poslední iteraci výpočtu pak provádí stejný počet vláken, jako je počet řádků vstupní matice.

```
__global__ void kernel_haarRows(const double *data, volatile
    double *output, const unsigned int width, const unsigned int
    rows, const unsigned int cols)
{
  const unsigned int col = getGlobalIdx_X() * 2; //expanze na každ
    ý druhý sloupec až do width * 2
  const unsigned int row = getGlobalIdx_Y();

  if (col >= width * 2 || row >= rows)
    return;
  const unsigned int a = col + row * cols;

  const unsigned int b = a + 1;

  const unsigned int indexPositive = col / 2 + row * cols;
  const unsigned int indexNegative = indexPositive + width;

  if (indexNegative >= cols * rows || b >= cols * rows)
    return;

  output[indexPositive] = (double)((data[a] + data[b]) / M_SQRT2);
  output[indexNegative] = (double)((data[a] - data[b]) / M_SQRT2);
}
```

Kód 5.2: Kernel vlnkové transformace řádků

Aby nedošlo k write-after-read hazardu, jsou výsledky v každé iteraci zapisovány do výstupního pole, jehož hodnoty jsou před následující iterací překopírovány na vstup. Jelikož každá další iterace pracuje pouze s první polovinou dat, jež byla získána z té předchozí, kopíruje se právě tato polovina.

```
__global__ void kernel_copyRows(volatile double *dst, const double
    *src, const unsigned int width, const unsigned int rows,
    const unsigned int cols)
{
  const unsigned int col = getGlobalIdx_X();
  const unsigned int row = getGlobalIdx_Y();
  if (col < cols && row < rows && col < width)
  {
    dst[col + row * cols] = src[col + row * cols];
  }
}
```

Kód 5.3: Kopírování změněných dat z výstupu na vstup

Transformace sloupců je prováděna analogicky – stačí zaměnit sloupce za řádky.

5.2.3 Extrakce význačných vlnek

K získání t význačných vlnek používám řadícího algoritmu implementovaného v knihovně thrust – potřebuji mezi vlnkami v podstatě najít jejich *t-tou* nejvyšší absolutní hodnotu. Ta se po seřazení dle absolutních hodnot bude nacházet na indexu t. Tuto hodnotu si uložím a v dalším kroku algoritmu Waveprint (generování sub-hashů) považuji každou nižší absolutní hodnotu za nulovou.

```
void inline Waveprint::extractTopWavelets(double *d_wavelets,
    size_t rows, size_t cols, int numToExtract, double *threshold)
{
  const unsigned int bytes = rows * cols * sizeof(double);

  cudaMemcpy(d_temporary, d_wavelets, bytes,
    cudaMemcpyDeviceToDevice);

  thrust::device_ptr<double> d_ptr(d_temporary);
  thrust::sort(d_ptr, d_ptr + cols * rows, sortAbs<double>());

  cudaMemcpy(threshold, &d_temporary[numToExtract], sizeof(double)
    , cudaMemcpyDeviceToHost);
  *threshold = fabs(*threshold);
}
```

Kód 5.4: Extrakce význačných vlnek

5.2.4 Tvorba sub-hashů

Sub-hash počítá kernel založený na paralelní redukci. Kernel je spouštěn na s blocích o 255 vláknech[19], kde s je délka jednoho sub-hashe.

[19] Maximální možná hodnota jednoho minhashe tak, aby bylo možné sloučit je po jednom bajtu. Viz kapitola 4, sekce 4.2.1.

Každý blok pracuje s jinou permutací dat získaných vlnkovou transformací. Každé vlákno bloku nejprve nalezne první nenulovou hodnotu pro jemu přidělenou část dat, kterou ukládá do pole ve sdílené paměti na pozici rovnou svému indexu. Pak jsou vlákna synchronizována a nad daty ve sdílené paměti je provedena paralelní redukce s operací minimum. Výsledek je uložen do výstupní proměnné, kterou je pole o velikosti *s*. Uložen je na pozici odpovídající indexu bloku. Redukce je optimalizována rozbalením na jednotlivé warpy.

```
extern __shared__ unsigned unsigned int s_nonZeroIndexes[];
const unsigned int threadId = threadIdx.x;
const unsigned int blockId = blockIdx.x;
s_nonZeroIndexes[threadId] = Waveprint::MAX_SUBHASH_VALUE;
__syncthreads();
if (blockId < signatureSize) {
  for (int j = threadId; j < Waveprint::MAX_SUBHASH_VALUE && j <
    sliceSize; j += blockSize) {
    if (fabs(wavelets[permutations[j + blockId * Waveprint::
    MAX_SUBHASH_VALUE]]) > threshold) {
      s_nonZeroIndexes[threadId] = j;
      break;
    }
  }__syncthreads();
  if (blockSize >= 1024) {
    if (threadId < 512)
      s_nonZeroIndexes[threadId] = min(s_nonZeroIndexes[threadId],
     s_nonZeroIndexes[threadId + 512]);
    __syncthreads();
  }
  if (blockSize >= 512) {
    if (threadId < 256)
      s_nonZeroIndexes[threadId] = min(s_nonZeroIndexes[threadId],
     s_nonZeroIndexes[threadId + 256]);
    __syncthreads();
  }
  if (blockSize >= 256) {
    if (threadId < 128)
      s_nonZeroIndexes[threadId] = min(s_nonZeroIndexes[threadId],
     s_nonZeroIndexes[threadId + 128]);
    __syncthreads();
  }
  if (blockSize >= 128) {
    if (threadId < 64)
      s_nonZeroIndexes[threadId] = min(s_nonZeroIndexes[threadId],
     s_nonZeroIndexes[threadId + 64]);
    __syncthreads();
  }
  if (threadId < 32)
    warpMinReduce<blockSize>(s_nonZeroIndexes, threadId);
  if (threadId == 0)
    hash[blockId] = s_nonZeroIndexes[0];
}
```

Kód 5.5: Vnitřek kernelu pro výpočet sub-hashů

V poli *permutations* jsou uloženy předem vygenerované permutace indexů pole vlnek.

5.2.5 Srovnání dvou otisků

Porovnání dvou otisků probíhá dle kapitoly 4, sekce 4.2.2. Z titulkového souboru je nejprve vytvořena databáze otisků tak, jak je popsáno v sekci 4.2.2. Ze zvukové stopy je pak vypočítána její signatura, která je následně rozdělena do l částí (sub-hashů).

Každá z částí je pak porovnána s vytvořenou databází. Ty, které získají více než h hlasů (viz 4.2.2), jsou společně s jim odpovídajícími titulky označeny za kandidátní sub-hashe.

```
std::map<int, SubhashCandidate> hashMatches;

//pro kazdou tabulku hashu
for (int i = 0; i < fmin(numOfHashTables, signatureSize); i++) {
  if (hashTables[i].find(signature[i]) == hashTables->end()) //
    neexistuje
    continue;

  auto matches = hashTables[i].equal_range(signature[i]);
  for (auto it = matches.first; it != matches.second; ++it) {
    int titleIndx = it->second.getIndex() - 1;
    if ((restrictIndex != SubtitleComparator::NO_MATCH_FOUND &&
    titleIndx != restrictIndex) || foundTitles[titleIndx])
      continue;
    else if (hashMatches.count(it->second.getIndex()))
      hashMatches.at(it->second.getIndex()).numOfMatchedTables++;
    else {
      SubhashCandidate candidate;
      candidate.numOfMatchedTables = 1;
      candidate.subhash = std::vector<unsigned long long>(
    signature, signature + signatureSize);
      candidate.subtitleIndex = it->second.getIndex();
      hashMatches.insert(std::pair<int, SubhashCandidate>(it->
    second.getIndex(), candidate));
    }
  }
}
//odstran indexy titulku, ktere maji mene nez threshold hlasu
for (auto it = hashMatches.begin(); it != hashMatches.end();) {
  if (it->second.numOfMatchedTables < threshold)
    it = hashMatches.erase(it);
  else
    it++;
}
return hashMatches;
```

Kód 5.6: Vnitřek funkce vyhledávající kandidáty

Pro kandidátní sub-hashe se stejnými titulky je vypočítána jejich Hammingova vzdálenost od celé signatury jejich titulku. Výsledek je vydělen délkou signatury. Pokud výsledná hodnota překročí předem určenou mez, je výsledek porovnání pozitivní.

```
unsigned int SubtitleMatcher::calculateHammingDistance(const std::
    vector<unsigned long long> &candidateHash, const std::vector<
    unsigned long long> &hash, unsigned int offset)
{
  double numOfMatches = 0;

  for (int i = 0; i < fmin(candidateHash.size(), hash.size() -
    offset); i++)
  {
    unsigned long long a = candidateHash[i];
    unsigned long long b = hash[i + offset];
    for (int j = 0; j < keysPerHash; j++)
    {
      //extrakce a porovnani puvodnich minhashu
      if ((a & 0xFF) == (b & 0xFF))
        numOfMatches++;

      a = a >> 8;
      b = b >> 8;
    }
  }
  return numOfMatches;
}
```

Kód 5.7: Výpočet Hammingovy vzdálenosti

5.3 Přehrávač videa

Při tvorbě přehrávače jsem postupoval podle návodu na [27] a dokumentace ke knihovně LibAV na [17].

Při přehrávání s titulky ve formátu fsrt je přehrávač spouštěn se třemi vlákny, která jsou tvořena a synchronizována pomocí knihovny SDL 2:

1. Vlákno dekódující obraz a zvuk
2. Vlákno porovnávající předem extrahované zvukové vzorky pomocí algoritmu Waveprint s titulky ve formátu fsrt
3. Hlavní, „zobrazovací“ vlákno

Na dekódujícím vlákně je pomocí funkcí knihovny LibAV dekódován obraz a zvuk filmu. Dekódované pakety jsou pak ukládány do front, ze kterých je postupně odebírá vlákno zobrazovací. Vzhledem k tomu, že pakety videa mohou přijít v jiném pořadí, než ve kterém mají být zobrazeny, jsou ukládány do

prioritní fronty. Ta je řazena dle času zobrazení, který každý paket obsahuje. Zvukové pakety přichází ve správném pořadí.

K zobrazení videa a přehrávání zvuku využívám knihovnu SDL 2.

Zobrazovací vlákno se cyklicky dotazuje nejprve na frontu s audio pakety. V případě, že není prázdná, je ze začátku fronty odebrán paket se zvukovými daty. Tento paket je následně pomocí funkcí knihovny SDL 2 předán zvukové kartě. Jakmile je zvukový paket předán zvukové kartě, získává ta nad ním kontrolu – přehrávání zvuku již není kontrolováno zobrazovacím vláknem.

Postup vykreslování videa je podobný, jen v jednu chvíli předávám k zobrazení pouze jeden snímek. Nový snímek zobrazuji jen ve chvíli, kdy je současný čas přehrávání (udaný zvukovou stopou) vyšší nebo roven času zobrazení paketu s tímto snímkem. Video je také nutné synchronizovat se zvukem: vykreslovací vlákno uspávám na čas, který po zobrazení relevantních videosnímků zbývá z doby trvání současného audio paketu (řádově jednotky až desítky milisekund).

Titulky jsou zobrazovány podobným způsobem jako video. Vlákno starající se o jejich časování je postupně ukládá do fronty, ze které jsou zobrazovacím vláknem postupně odebírány a ukládány do lokální fronty. Jakmile pak přehrávání dojde k času, ve kterém má být následující titulek ukázán, je zobrazen. Stejným způsobem probíhá skrývání titulků při příchodu času jejich skrytí.

5.4 Použití programu

Veškeré parametry Waveprint algoritmu jsou nastaveny napevno na hodnoty, které byly zjištěny autory v [7], či mnou v prvotních testech.

Samotný program přijímá až tři parametry, v tomto pořadí:

1. Cesta k souboru videa
2. Cesta k prvnímu souboru s titulky
3. Cesta k druhému souboru s titulky

Pro konverzi mezi dvěma titulkovými formáty je nutné předat všechny tři argumenty. První soubor s titulky je pak zdrojem a jeho přípona (*.srt nebo *.fsrt) udává, z jakého formátu se konvertuje. Ukázku průběhu konverze lze nalézt na obrázku 5.1.

Vynecháním třetího parametru dojde k přehrání videa s titulky ve formátu předaného titulkového souboru (dle přípony).

Pro přehrání samotného videa stačí předat samotný parametr s cestou k souboru s videem.

Před přehráváním či konverzí jsou ze souboru videa vždy extrahovány vzorky audio stopy, které jsou poté převzorkovány do potřebného formátu (mono, 2kHz). Tyto vzorky jsou pak použity v algoritmu Waveprint jako

vstupní data. Jejich předzpracováním šetřím výpočetní výkon při samotném přehrávání/konverzi mezi titulkovými formáty.

```
PS G:\škola\diplomka\vs> .\diplomka_cuda.exe .\hitchhiker.mp4 .\hitchhiker_test.fsrt .\test.srt
Extracting audio samples. Please wait...
Done extracting audio samples.
Done loading fingerprinted subtitles.
Computing timestamps from fingerprinted subtitles...
Fingerprint of samples at 30600 is 100.00% similar with start fingerprint of subtitle 1
00:00:30,600 --> 00:00:38,600
<i>It's an important and popular fact</i>


End found at 33725 ms               {0.52 %}                                ]
1
00:00:30,600 --> 00:00:33,725
<i>It's an important and popular fact</i>


Fingerprint of samples at 33775 is 100.00% similar with start fingerprint of subtitle 2
00:00:33,775 --> 00:00:41,775
<i>that things are not always
what they seem.</i>


End found at 37650 ms               {0.58 %}                                ]
2
00:00:33,775 --> 00:00:37,650
<i>that things are not always
what they seem.</i>


Fingerprint of samples at 37725 is 100.00% similar with start fingerprint of subtitle 3
00:00:37,725 --> 00:00:45,725
<i>For instance, on the planet Earth,
man had always assumed</i>


End found at 41375 ms               {0.63 %}                                ]
3
00:00:37,725 --> 00:00:41,375
<i>For instance, on the planet Earth,
man had always assumed</i>
```

Obrázek 5.1: Ukázka použití programu: konverze z formátu fsrt do srt.

Při konverzi do *fsrt* mají titulky ve výsledném souboru podobný tvar, jako je tomu na obrázku 5.2.

```
4
41450_45275
16777472;68487684;84216344;234892039;218958345;167903744;1512991;16908546;134480135;84087314;50792966;470483459;453577743;68028166;839523
88;218169345;100794372;1508875;17170433;134547220;17240602;154015501;117574925;83951873;134481408;17760800;67247656;469958670;604374284;6
02113;402786562;17575473;16993809;184820489;318832641;103678216;151194405;666625;68027914;184683270;268764174;337709587;117507845;2357370
95;252510209;50659588;19006218;17367830;34341121;252255237;16910603;69609225;301989910;151392788;302983429;470220834;528650;16785665;1737
8846;318973441;251789581;252051971;455344900;68420884;137803;419828997;33623554;121767945;252447758;251726862;486936072;202051084;2189657
75;741998612;655688478;186194193;68955924;939787278;336405816;85197057;469897475;16978177;19803419;420221745;33620253;772217345;1250563;2
35670534;1026033184;84938008;335942153;34755347;395023;18290446;5838856;605053698;117777668;168166926;393485;537661713 -->
16778513;17045252;235078943;150995990;102238985;370147336;76289;16919298;117702960;201397779;13;470024707;135267844;68683531;83953461;504
62739;268763137;117506315;705432579;33751307;218890499;134614272;453836807;83951873;503582976;402721056;69010436;50855948;151195414;59624
6;738329345;167782934;603979785;34473484;83955713;198152;470485763;596751;906036745;50466065;134285846;185861126;266521;253756163;2519859
21;251658503;16843522;17042484;168689665;251665920;16910603;67635211;526712856;151392774;302984193;470223138;17307153;16777498;18612224;6
7184129;52042253;252641795;571019523;67371035;148236;419573551;487075598;117507356;252447246;202834200;201525505;1174670342;33699619;3207
33203;18156552;419629330;72622868;219022345;151865622;85205547;469899792;538711847;17048091;135285778;436408325;121441281;823334912;23901
9777;19138080;84020774;453380873;504498707;121767685;18296845;336006152;454038274;101019711;170596615;104071444;117441288
<i>that he was the most intelligent
species occupying the planet,</i>
```

Obrázek 5.2: Podoba titulku ve formátu fsrt.

Pro zobrazení nápovědy spusťte program pouze s přepínačem *-h* nebo *–help*.

Kapitola 6

Testování

V této kapitole jsou uvedeny výsledky měření kvality nasazení titulků v závislosti na změnách v příslušném filmu.

Veškerá měření byla prováděna na stroji s parametry uvedenými v tabulce 6.1.

Tabulka 6.1: Parametry testovacího stroje

OS	Windows 10
CPU	Intel Core i5-3570K
GPU	GeForce GTX 1060 6 GB
RAM	DDR 3, 16 GB
Verze CUDA	9.0

Testy jsem prováděl na anglické verzi filmu Stopařův průvodce po galaxii s anglickými titulky s celkovým počtem 1382 titulků a na anglické verzi filmu Dokonalý trik s českými titulky o celkovém počtu 1533 titulků.

Veškeré změny ve zvukovém signálu byly provedeny pomocí programu Audacity verze 2.2.1 [31].

Abych otestoval účinnost samostatného algoritmu Waveprint pro použití při časování titulků, zatížil jsem hashe titulků vahou 1.0 a časovou složku vahou 0.0[20].

6.1 Zvuková stopa beze změn

Nejprve jsem otestoval kvalitu titulků při přehrávání se stejnou verzí filmu, na kterou byly nasazeny. Jak se dalo předpokládat, v tomto základním testu jsou výsledky dobré. U Stopařova průvodce po galaxii bylo z celkových 1382 titulků

[20] Viz kapitola 4, sekce 4.1

nalezeno 1368, z toho u 12 bylo nutné ukončit hledání[21]. Většina z těchto ukončených titulků byla zároveň falešnými pozitivy. Těch bylo celkem 35.

Výsledky filmu Dokonalý trik jsou podobné. Z celkového počtu 1533 titulků jich bylo 1525 nalezeno, z toho u 12 bylo nutné ukončit hledání. Počet falešných pozitiv je 12.

Falešná pozitiva jsou také odpovědná za to, že průměrná jistota v nalezené titulky není stoprocentní – u falešných pozitiv byla dokonce někdy menší než 60 %. Vyšší průmerná jistota v nalezený titulek u Dokonalého triku pak odpovídá menšímu počtu falešných pozitiv. Tento nižší počet falešných pozitiv si vysvětlují ne příliš výraznou hudbou ve filmu. Naopak ve Stopařově průvodci po galaxii je hudba na pozadí výrazná.

Tabulka 6.2: Zvuk beze změn

	Průměrná jistota v nalezený titulek	**Procento nalezených titulků**	**Počet falešných pozitiv**
Stopařův průvodce po galaxii	98.7 %	99.0 %	35
Dokonalý trik	99.7 %	99.5 %	12

6.2 Zvuková stopa s přidanou slabou ozvěnou

Pro druhý test jsem pomocí programu Audacity do původní zvukové stopy přidal slabou ozvěnu: čas zpoždění byl nastaven na 1 vteřinu s faktorem doběhu 0,1.

Tabulka 6.3: Zvuk s přidanou slabou ozvěnou

	Průměrná jistota v nalezený titulek	**Procento nalezených titulků**	**Počet falešných pozitiv**
Stopařův průvodce po galaxii	82.6 %	98.6 %	36
Dokonalý trik	88.8 %	99.3 %	12

V tomto testu ozvěna ještě výsledky hledání titulků příliš neovlivnila. Počety nalezených titulků i počety falešných pozitiv jsou u obou testovaných filmů srovnatelné s testem audio stopy beze změny. Snížily se průměrné jistoty v nalezený titulek, podle počtů falešných pozitiv však toto nijak jejich kvalitu neovlivnilo.

[21] Byly delší než 8 vteřin, viz kapitola 5, sekce 5.1

6.3 Zvuková stopa s přidanou výraznou ozvěnou

V tomto testu jsem postupoval stejně jako u testu předchozího, jen parametry použité ke generování ozvěny byly změněny. Čas zpoždění byl nastaven na 2 vteřiny, čas doběhu 0,4.

Tabulka 6.4: Zvuk s přidanou výraznou ozvěnou

	Průměrná jistota v nalezený titulek	**Procento nalezených titulků**	**Počet falešných pozitiv**
Stopařův průvodce po galaxii	65.8 %	40.1 %	41
Dokonalý trik	66.1 %	45.1 %	18

Zde byla ozvěna dosti znatelná – v dialozích bylo mnohdy těžší lidským uchem odlišit ozvěnu od slova. Jak je vidět v tabulce 6.4, tato změna významně ovlivnila i efektivitu programu při rozpoznávání titulků.

Ve filmu Stopařův průvodce po galaxii bylo nalezeno pouze 558 titulků z původních 1382. Také jistota v titulky, které nalezeny byly, je nižší – pouze 65,8 %.

Jistota v nalezené titulky je u Dokonalého triku ovlivněna o něco méně, což si vysvětluji tím, že hlasy jsou v tomto filmu méně výrazné, než ve Stopařově průvodci po galaxiil. A vliv ozvěny je znát především v dialozích. Pokles počtu nalezených titulků je ale stále výrazný: z celkových 1533 jich bylo nalezeno 691. Jistota v nalezený titulek je srovnatelná se Stopařovým průvodcem po galaxii – 66.1 %.

Počet titulků, k nimž nebyl nalezen koncový čas, je také značný: v obou filmech se ho nepodařilo nalézt u téměř 33 % z nalezených titulků. Na vině jsou částečně falešná pozitiva (především u Stopařova průvodce po galaxii), částečně také ozvěnou silně snížená kvalita vstupního signálu.

6.4 Zvuková stopa s přidaným slabým bílým šumem

Bílý šum byl do původní zvukové stopy namixován pomocí programu Audacity. Parametr rozkmitu byl nastaven na 0,1. Šum byl poté zeslaben na -18 dB tak, aby byl uchem jen steží rozpoznatelný.

Výsledky jsou téměř shodné s testem původní zvukové stopy. Jen se o pár několik procent snížila jistota v nalezené titulky.

Tabulka 6.5: Zvuk s přidaným slabým bílým šumem

	Průměrná jistota v nalezený titulek	**Procento nalezených titulků**	**Počet falešných pozitiv**
Stopařův průvodce po galaxii	95.3 %	99.0 %	35
Dokonalý trik	93.3 %	99.3 %	10

6.5 Zvuková stopa s přidaným výrazným bílým šumem

Bílý šum byl v tomto testu do původní audio stopy přidán stejným způsobem jako v předchozím testu. Zeslaben byl pouze na -5 dB. Šum byl pak lidským uchem jednoduše registrovatelný. U Dokonalého triku pak byly s tímto nastavením hlasy jednotlivých postav mnohem hůře rozpoznatelné, než tomu bylo u Stopařova průvodce po galaxii.

Tabulka 6.6: Zvuk s přidaným výrazným bílým šumem

	Průměrná jistota v nalezený titulek	**Procento nalezených titulků**	**Počet falešných pozitiv**
Stopařův průvodce po galaxii	91.8 %	98.5 %	35
Dokonalý trik	74.2 %	81.9 %	10

Výsledky testu Stopařova průvodce po galaxii jsou o poznání lepší, než je tomu u Dokonalého triku. To si vysvětluji tím, že dialogy jsou v Dokonalém triku méně hlasité. Jsou pak více přehlušeny šumem. I přesto jsou však výsledky dobré.

Podle výsledků lze tedy tvrdit, že přidání bílého šumu efektivnost časování titulků příliš nepoškozuje, pokud nepřehluší původní zvukovou stopu. To ani není vzhledem k povaze algoritmu Waveprint překvapující: signatura je tvořena jen z úzkého frekvenčního spektra, čímž je frekvenční spektrum šumu ořezáno. Algoritmus dále těží z faktu, že pro tvorbu sub-hashů vybíráme t nejvýznačnějších vlnek – rozumná úroveň se na těch nejcharakterističtějších projeví minimálně.

Problém pak představuje šum, který přehluší původní zvuk filmu, především dialogy. Takto znehodnocená zvuková stopa by však představovala problém i pro lidské ucho.

Proto je také vidět rozdíl v úspěšnosti programu u obou filmů. V Dokonalém triku nejsou především hlasy tak výrazné, jako je tomu u Stopařova průvodce po galaxii.

6.6 Střih ve filmu

V tomto testu jsem vyzkoušel funkčnost programu při použití na jinou verzi filmu – například pokud mám titulky na režisérský sestřih a chci je použít na verzi, ve které některé scény chybí.

Proto jsem u obou testovaných filmů provedl ve zvukové stopě 10 minut dlouhý střih mezi 20. a 30. minutou.

Změnil se zde celkový počet titulků. U Stopařova průvodce po galaxii jich z 1382 po střihu zbylo 1252. V Dokonalém triku z původních 1533 zbylo 1417 titulků.

Tabulka 6.7: Střih ve filmu

	Průměrná jistota v nalezený titulek	**Procento nalezených titulků**	**Počet falešných pozitiv**
Stopařův průvodce po galaxii	77.5 %	82.0 %	49
Dokonalý trik	98.6 %	99.2 %	6

Co se filmu Stopařův průvodce po galaxii týče, výsledky jsou horší, než u původní verze zvukové stopy. Počet nalezených titulků a jistota v n+ je nižší, kdežto počet falešných pozitiv se mírně zvýšil. Několik z falešných pozitiv vzniklo chybným přiřazením titulků z vystřiženého úseku k jiným částem filmu.

Snížená jistota v nalezené titulky i snížený počet titulků nalezených jsou pak zaviněny částečně falešnými pozitivy, částečně nepříznivým zarovnáním zvuku po provedeném střihu. Výsledky však stále považuji za uspokojivé.

Na druhou stranu výsledek testu filmu Dokonalý trik je srovnatelný s testem jeho původní zvukové stopy. Zde totiž řezem ve filmu nedošlo k posunu zarovnání zvukové stopy.

U tohoto filmu zde dokonce zmizela některá falešná pozitiva – ta, která vzikla právě v onom odstraněném úseku.

6.7 Shrnutí

V následující tabulce jsou všechna předchozí měření zprůměrována:

Jak je z naměřených dat vidět, největší problém představuje výrazná ozvěna ve zvukové stopě – zdrojový signál je při ní příliš znehodnocen. Při tvorbě signatur vycházím ze spektrogramů v různých časových úsecích[22]. Pokud do jednotlivých úseků zanesu dostatečně silné nové frekvence, povaha spektrogramů se výrazně změní, což se promítne do výsledné signatury.

[22]Pomocí STFT, viz sekce 5.2.1

Tabulka 6.8: Shrnutí měření

	Průměrná jistota v nalezený titulek	**Průměrné procento nalezených titulků**	**Průmerný počet falešných pozitiv**
Původní signál	99.2 %	99.3 %	23.5
Slabá ozvěna	85.7 %	99.0 %	24
Výrazná ozvěna	66.0 %	42.6 %	29.5
Slabý bílý šum	94.3 %	99.2 %	22.5
Výrazný bílý šum	83 %	90.2 %	22.5
Střih ve filmu	88.6 %	90.6 %	27.5

Přidání rozumné úrovně šumu funkčnost programu výrazně neovlivnilo – výsledky jsou stále srovnatelné s výsledky původní audio stopy. Zde velmi pomáhá použití pouze úzkého frekvenčního spektra. Avšak pokud šum začíná zvukové stopě dominovat, rozlišovací schopnost algoritmu se snižuje. To je však očekávatelné.

Výsledky při střihu ve filmu jsou uspokojivé, ukazují však určitý problém při nepřesném zarovnání zvukové stopy po střihu. Tento problém by vyřešilo použití delších zvukových úseků při časování titulků. Pak by ale již nebylo možné titulky časovat souběžně s přehráváním videa.

Kapitola 7

Závěr

7.1 Analýza možnosti použití souběžně s filmovou projekcí

Titulkový formát v současné podobě není možné použít souběžně s filmovou projekcí.

První problém je ten, že má implementace počítá se zvukovými vzorky předem uloženými do operační paměti. Při současné projekci by data přicházela v reálném čase. Nejedná se však o velkou překážku, se současným nastavením dokáže stroj s parametry alespoň dle tabulky 6.1 titulky nasazovat v reálném čase.

Další překážka se v mé implementaci vyskytuje u tvorby spektrogramů ze zvuku v časovém úseku, který končí až po začátku/konci titulku[23]. Při 5-ti vteřinových úsecích, jejichž střed je nastaven přesně na začátek, respektive konec titulku, by výsledkem bylo 2,5 vteřinové zpoždění titulků. Řešením tohoto problému by bylo posunutí těchto 5-ti vteřinových úseků tak, aby končily přesně v bodě, pro který chci titulek časovat. Problém tohoto přístupu však může být vyšší výskyt falešných pozitiv v místech se shodnou zvukovou stopou – například ve scénách s nízkou frekvencí dialogů (a tedy titulků) a stejnou hudební skladbou z filmového soundtracku na pozadí. Částečná inkluze dialogových částí přidávala takovým zdrojovým datům na unikátnosti.

Tento problém by byl pravděpodobně vyřešen použitím delších úseků zvukové stopy (30 nebo 60 vteřin) pro tvorbu otisků. To by však mělo za následek výrazný nárůst složitosti výpočtu – titulky by nebylo možné nasazovat v reálném čase.

Jako další řešení se nabízí dát při rozhodování o přijetí titulku nižší váhu signaturám získaným ze zvukové stopy, a naopak dát určitý vliv časovému kódu, který se ve formátu fsrt nachází také[24]. Zde bychom však narazili na

[23] Viz kapitola 5, sekce 5.2.1

[24] Viz kapitola 4, sekce 4.1

problém v případě střihu ve filmu – po střihu by váhy časového kódu snižovaly pravděpodobnost přijetí správného titulku.

Převedení zbytku algoritmu Waveprint, tedy včetně porovnávání signatur, na GPU může také pomoci.

Problémem je také výskyt falešných pozitiv. A to i v případě přehrávání titulků společně s verzí filmu, na kterou byly nasazeny – je zde sice méně častý, ideálně by však měl být pro co nejlepší divácký prožitek nulový. Jako řešení se opět nabízí použít delší úseky zvukové stopy při tvorbě signatur. To by však, jak již bylo řečeno, znemožnilo časování titulků v reálném čase.

7.2 Shrnutí a návrh vylepšení

V práci jsem provedl analýzu současných titulkových formátů a problémů s jejich časováním. Tyto problémy vyvstávají z faktu, že běžné titulkové formáty spoléhají na přehrávání se správnou verzí filmu. Například u sestříhané verze filmu pak titulky na verzi nesestříhanou nesedí. Práce na [5] se sice zabývá dynamickým seřizováním titulků pomocí otisků zvukové stopy, neřeší však problém vyvstávající při výskytu střihu ve filmu.

Navrhl jsem titulkový formát *fsrt*, ve kterém jsou titulky časovány primárně pomocí otisků zvukové stopy. Ty získávám pomocí algoritmu Waveprint, který k zmenšení velikosti jednotlivých signatur používá vlnkové transformace a techniky minhashingu [7].

Část algoritmu Waveprint, která signatury generuje, jsem implementoval pomocí CUDA. Z časových důvodů jsem se však rozhodl CUDA při implementaci porovnávání dvou signatur nepoužít. To se nabízí zařadit mezi možná vylepšení.

V budoucnu by bylo zajímavé zjistit, jak by byla efektivita v hledání shodných signatur ovlivněna, kdyby byl z algoritmu Waveprint odstraněn krok tvorby spektrogramů pomocí STFT – vzorky zvukových dat by byly rozděleny a poté by na ně byla rovnou aplikována Haarova vlnková transformace.

Součástí implementace je jednoduchý přehrávač, který umožňuje titulky ve formátu *fsrt* časovat souběžně s přehráváním videa. správné zobrazení titulků je však závislé také na výkonu stroje, na kterém je film přehráván: při nedostatečném výkonu film titulky „předběhne“. Řešením, které jsem částečně také z časových důvodů nepoužil, by bylo film při prázdné titulkové frontě zastavit a čekat na další titulky. A divácký prožitek také není u filmu, který je neustále (nebo dlouze) bufferován, nijak veliký. Zároveň by mohl vyvstat problém při sestříhaném filmu. Pokud by byly již všechny relevantní titulky přehrány, ale přebývaly by ty, které původně patřily do vystřižených scén, pak by se přehrávání zastavilo při čekání na titulky, které nepřijdou.

Přehrávač videa by bylo také vhodné doplnit o funkci vyhledávání v přehrávaném filmu, což jsem z časových důvodů neučinil.

V současné podobě není mou implementaci možné použít souběžně s filmovou projekcí – důvody a z nich vyplývající nutné změny jsou uvedeny v sekci 7.1. Především je nutné urychlit či zefektivnit výpočet a porovnávání zvukových signatur. Toho lze dosáhnout buď spuštěním na silnějším stroji, nebo doimplementováním celého algoritmu pomocí CUDA. Některé části mé implementace lze také zefektivnit. Například extrakci význačných vlnek, kterou implementuji pomocí řazení[25], by bylo vhodnější implementovat pomocí výběrového algoritmu – například QuickSelect.

Práce však přesto ukazuje, že algoritmus Waveprint je možné použít pro časování titulků pomocí otisků zvukové stopy. Má implementace si uspokojivě poradila se střihem v první čtvrtině filmu, a tedy nutností přečasovat většinu následujících titulků s vynecháním těch chybějících. Zároveň se ukazuje, že algoritmus Waveprint funguje i při použití krátkých úseků zvukové stopy.

[25] Viz kapitola 5, sekce 5.2.3

Literatura

[1] Foto projekce elektronických titulků z FebioFestu. [online], [cit. 2017-11-11]. Dostupné z: http://portlingua.cz/fotogalerie.aspx

[2] Pošta, M.: *Titulkujeme profesionálně*. Praha: Apostrof, druhé vydání, 2012, ISBN 978-80-87561-16-4.

[3] Griffin, E.: Caption Format Acronyms, Explained. březen 2015, [cit. 2017-10-14]. Dostupné z: http://www.3playmedia.com/2015/03/05/caption-format-acronyms-explained/

[4] Staša, M.: Formát SUB (MicroDVD). duben 2013, [cit. 2017-11-01]. Dostupné z: https://www.titulkovani.cz/o-titulkovani/index.php?blog=format-sub-microdvd

[5] C. Villa Real, L.; Laiola Guimarães, R.; Avegliano, P.: Dynamic Adjustment of Subtitles Using Audio Fingerprints. *Conference: Proceedings of the 23rd ACM international conference on Multimedia, At Brisbane, Australia*, říjen 2015, [cit. 2017-09-29]. Dostupné z: https://www.researchgate.net/publication/291348353_Dynamic_Adjustment_of_Subtitles_Using_Audio_Fingerprints

[6] DiCola, T.: FFT: Fun with Fourier Transforms. [online], 2015, [cit. 2017-12-01]. Dostupné z: https://learn.adafruit.com/fft-fun-with-fourier-transforms/background

[7] Shumeet Baluja, M. C.: Content Fingerprinting Using Wavelets. *The 3rd European Conference on Visual Media Production (CVMP 2006)*, listopad 2006, [cit. 2017-09-29]. Dostupné z: http://ieeexplore.ieee.org/document/4156050/

[8] Statut 50. MFF Karlovy Vary (3. - 11. července 2015). 2015, [cit. 2017-11-09]. Dostupné z: http://www.kviff.com/docs/Statut_50_MFFKV.pdf

[9] Wanková, K.: *Titulkování živě přenášených divadelních představení.* diplomová práce, Univerzita Karlova, 2014.

[10] Speechpad: *SMPTE-TT (Society of Motion Picture and Television Engineers Timed Text).* 2017, [cit. 2017-10-14]. Dostupné z: https://www.speechpad.com/captions/smpte-tt

[11] Timed Text Format (SMPTE-TT). prosinec 2010, [cit. 2017-10-22]. Dostupné z: https://www.smpte.org/sites/default/files/st2052-1-2010.pdf

[12] CEA-708. 2017, [cit. 2017-10-22]. Dostupné z: http://www.theclosedcaptioningproject.com/?p=389

[13] Alexander, P.: Elektronické titulky pro sluchově postižené do kin. červenec 2014.

[14] Selcuk Aslan, D. K., Hasan Badem: Accelerating Discrete Haar Wavelet Transform on GPU cluster. *2015 9th International Conference on Electrical and Electronics Engineering (ELECO)*, listopad 2015, [cit. 2017-11-24]. Dostupné z: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=7394516

[15] Addison, P. S.: *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance.* CRC Press, první vydání, 2017, ISBN 9781315372556.

[16] Cuff, P.: FFT and Spectrogram. [online], 2015, [cit. 2017-11-24]. Dostupné z: https://www.princeton.edu/~cuff/ele201/files/spectrogram.pdf

[17] Libav team: *Libav Documentation.* [cit. 2017-11-25]. Dostupné z: https://libav.org/documentation/doxygen/master/

[18] Simple Directmedia Layer. [cit. 2017-11-26]. Dostupné z: https://www.libsdl.org/

[19] CUDA. [online], 2017, [cit. 2017-12-01]. Dostupné z: https://developer.nvidia.com/cuda-zone

[20] CUDA C Programming Guide. [online], 2017, [cit. 2017-12-07]. Dostupné z: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[21] Thrust. [online], září 2017, [cit. 2017-11-27]. Dostupné z: http://docs.nvidia.com/cuda/thrust/index.html

[22] cuFFT. [online], září 2017, [cit. 2017-11-27]. Dostupné z: http://docs.nvidia.com/cuda/cufft/index.html

[23] Charles E. Jacobs, D. H. S., Adam Finkelstein: Fast Multiresolution Image Querying. leden 1995, [cit. 2017-10-05]. Dostupné z: https://pdfs.semanticscholar.org/fd46/cb7dafe910127d2407d28fa7ac8aabddc3ce.pdf

[24] Mullen, L.: Minhash and locality-sensitive hashing. listopad 2016, [cit. 2017-11-19]. Dostupné z: https://cran.r-project.org/web/packages/textreuse/vignettes/textreuse-minhash.html

[25] Shumeet Baluja, M. C.: Known-Audio Detection using Waveprint: Spectrogram Fingerprinting by Wavelet Hashing. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '07)*, duben 2007, [cit. 2017-12-08]. Dostupné z: http://ieeexplore.ieee.org/document/4217060/

[26] Shumeet Baluja, M. C.: Audio Fingerprinting: Combining Computer Vision and Data Stream Processing. *2007 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '07)*, duben 2007, [cit. 2017-11-21]. Dostupné z: http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=4217383

[27] Martin Bohme, F. B.: An ffmpeg and SDL Tutorial or How to Write a Video Player in Less Than 1000 Lines. 2015, [cit. 2017-11-25]. Dostupné z: https://github.com/haasn/mpvhq-old/wiki/FFmpeg-versus-Libav

[28] Hsueh-szu Yang, B. K.: Time Stamp Synchronization in Video Systems. 2010, [cit. 2017-11-27]. Dostupné z: http://www.ttcdas.com/pdf2/tp_2010_time_synch_video.pdf

[29] Tarantola, A.: Why Frame Rate Matters. leden 2015, [cit. 2017-11-26]. Dostupné z: https://gizmodo.com/why-frame-rate-matters-1675153198

[30] Gutierrez-Osuna, R.: L6: Short-time Fourier analysis and synthesis. [online], 2002, [cit. 2017-12-27]. Dostupné z: http://research.cs.tamu.edu/prism/lectures/sp/l6.pdf

[31] Team, A.: AUDACITY®: Free Audio Editor and Recorder, verze 2.2.1 [software]. [online], 2017, [cit. 2017-12-30]. Dostupné z: http://www.audacityteam.org/

Příloha **A**

Seznam použitých zkratek

CPU Central Processing Unit. 13

CUDA Compute Unified Device Architecture. 3

FFT Fast Fourier Transform. 13

GPU Graphics Processing Unit. 13

OS Operating System. 31

RAM Random Access Memory. 31

SDK Software Development Kit. 13

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu přiloženého CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src` .. adresář obsahující zdrojové kódy v práci diskutovaného programu a tohoto textu
  - `impl` ..... zdrojové kódy implementace v jazyce C++
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... adresář obsahující text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS