FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Detekce statických snímků a animací ze záznamu obrazovky
Student:	Richard Burkoň
Vedoucí:	Ing. Petr Pulc
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Záznam převážně statické obrazovky počítače s použitím tradičních kompresních algoritmů je značně neefektivní. I pokud kompresní algoritmus uvažuje rozdílovou informaci mezi snímky, klíčové snímky uchovávají informaci kompletní. Snižování počtu snímků za vteřinu nebo prodlužování vzdálenosti mezi klíčovými snímky pak vede k degradaci animací promítaných na obrazovce.

Cílem práce je navrhnout systém, který pro ukončený záznam obrazovky vytvoří sadu statických snímků a videosouborů, a soubor s metadaty o umístění těchto souborů na časové ose. V případě obvykle používané digitální cesty signálu je možné zanedbat i přítomnost šumu nebo jiných artefaktů.

Pro splnění zadání je nezbytné, aby student:
- prozkoumal současná řešení detekce nového statického snímku
- navrhl vlastní metodu rozlišení statického snímku a animace s omezenou pamětí
- metodu implementoval s pomocí knihovny OpenCV
- metodu vhodně otestoval.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Detekce statických snímků a animací ze záznamu obrazovky

Richard Burkoň

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petr Pulc

28. června 2018

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 28. června 2018 ……

Odkaz na tuto práci

Burkoň, Richard. *Detekce statických snímků a animací ze záznamu obrazovky.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Bakalářská práce se zabývá zpracováním záznamu prezentací. Výsledný program analyzuje kontinuální záznam obrazovky s jednotlivými slidy prezentace, který byl pořízen například pomocí programů třetích stran pro záznam obrazovky nebo nahráním signálu obrazovky do externího rekordéru. Výstupem programu je prezentace převedená do formátu statických snímků s jednotlivými slidy prezentace spolu s časovými značkami, kdy došlo ke změně slidu. Součástí je i podpora rozpoznávání videa nebo animace v prezentaci, které ukládá jako samostatné video soubory. Program je psán v jazyce C++ s podporou knihovny OpenCV, která umožňuje základní manipulaci s videi a obrazy.

Klíčová slova analýza videa, přednáška, detekce slidů, porovnávání snímků, opencv, c++

Abstract

The topic of this work is processing records of lecture presentations. The program analyses the screen capture with presentation slides recorded with third-party programs or by recording the video signal with an external recorder. This capture is converted into a series of static slides and video files with individual slides and animations from the presentation. The file with timestamps of slides change is also included. The program is written in C++ with the support of the OpenCV library, which allows basic manipulation with videos and images.

Keywords video analysis, lecture, slides detection, image comparison, opencv, c++

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Pořizování videí z různých akcí je v poslední době jedním z velkých trendů, proto se jí nevyhýbá ani oblast nahrávání přednášek. V dnešní době se ale nemusí, i díky pokrokům v oblasti přehrávání videa ve webových prohlížečích, zaznamenané přednášky ukládat jen na běžné portály pro sdílení videa (například YouTube). Na tyto portály je kvůli omezení na přehrávání pouze jednoho video souboru potřeba umístit záznam přednášejícího i samostatných slidů s prezentací do jednoho videa. Toto rozložení se nazývá Picture-in-picture a mezi jeho hlavní nevýhody patří, že jeden ze vstupních souborů (nebo oba) musí být zmenšeny na úkor druhého, případně se musí obrazové signály překrývat. Pokud prezentace obsahuje malé písmo nebo přednášející ukazuje nějaký předmět, může dojít k okamžiku, kdy divák nepochytí všechen obsah.

Tuto problematiku řeší například platforma SlidesLive, která umožňuje nezávisle vedle sebe umístit video a prezentaci s jednotlivými slidy, přičemž divák může prezentaci posouvat nezávisle na videu a naopak. Záznam přednášek pro tuto platformu probíhá pomocí již vytvořeného speciálního programu, který vytvoří statický snímek všech obrazovek připojených k danému počítači v momentě, kdy dojde ke stisknutí klávesy, tlačítka myši nebo prezenteru. Pokud však aplikaci nelze spustit na počítači přednášejícího (nekompatibilnost OS, nastavená bezpečností opatření, která neumožňují spouštět cizí software, ...), je nutné prezentaci nahrát na externí video rekordér a poté ze záznamu manuálně vystříhat statické snímky a videa a najít k nim okamžiky, kdy došlo ke změně, což je značně časově náročné. Další možností je využití platformy Mediasite od Sonic Foundry, která řeší nahrávání přednášek pomocí snímací karty v kombinaci s dodávaným softwarem, který je schopnen ve vstupním analogovém VGA signálu detekovat změny snímků. Bohužel kvůli vysokým cenám a nemožnosti kombinace s komponentami jiných výrobců je tato platforma pro běžného uživatele nepoužitelná.

Z výše uvedených důvodů jsem se rozhodl pro volbu tématu automatické detekce slidů z video souboru se záznamem obrazovky. Vytvořený program by

měl analyzovat video soubor se záznamem slidů a pomocí porovnání jednotlivých snímků ve videu nacházet okamžiky, kdy došlo ke změně slidu. Samostatné slidy z videa bude ukládat jako PNG obrázky a čas změny zaznamenávat do XML souboru. Díky tomu může výsledný program pomoci lidem, kteří se zabývají zpracováním přednášek. Kromě statických snímků by měl ve videu detekovat i animace a vložená videa, která bude ukládat jako samostatné video soubory.

V následující kapitole se budu zabývat analýzou problému a souhrnem současných alternativních řešení. Dále v 2.2 je v práci analyzováno několik různých metod pro porovnání dvou snímků a zjištění jejich podobnosti. V kapitole 2.3 jsou všechny metody porovnány a shrnuty jejich výhody a nevýhody. Na závěr v kapitole 3 se věnuji samotné implementaci zmíněných metod pro porovnávání dvou snímků v jazyce C++ s využitím knihovny OpenCV.

Kapitola 1

Cíl práce

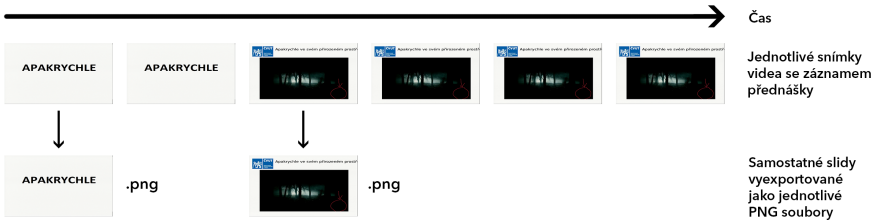
Cílem bakalářské práce je vytvořit program, který bude automaticky detekovat přechody mezi jednotlivými slidy v ukončeném záznamu obrazovky s prezentací. Jednotlivé slidy uloží buď ve formátu PNG, pokud se bude jednat o statické slidy, nebo do souboru MP4 pro animace/video slidy. Součástí výstupu bude i XML soubor, který bude obsahovat časové značky, kdy došlo ke změně slidu, aby bylo možné záznam obrazovky zpětně zrekonstruovat.

V teoretické části je provedena analýza různých algoritmů pro rozpoznání rozdílnosti mezi dvěma statickými snímky. V ní je vysvětleno, jak jednotlivé algoritmy pracují, a též je zde provedeno porovnání výhod a nevýhod jednotlivých algoritmů. Na konci teoretické části je shrnuto, který algoritmus je nejvýhodnější pro porovnání dvou statických snímků z různých druhů/typů prezentací.

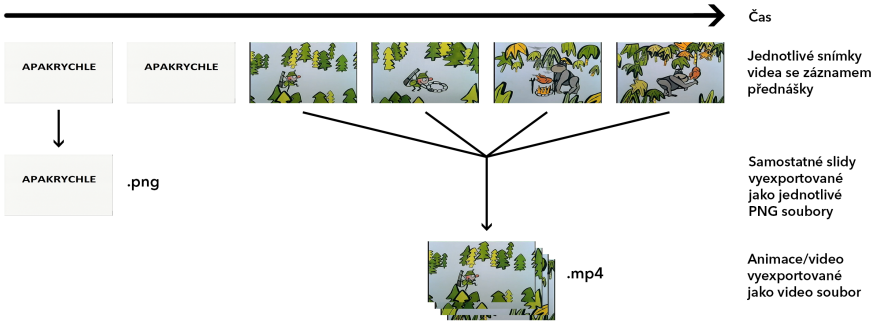
V praktické části se věnuji samotné implementaci programu pro detekci slidů. Program obsahuje implementaci zmíněných metod z teoretické části a uživatel si tak může sám zvolit, který algoritmus pro porovnání využije.

Požadovanou funkci bakalářské práce je možné si názorně přiblížit na obrázku 1.1, kde je znázorněna sekvence snímků ve videu. Algoritmus tyto snímky prochází a pokud detekuje změnu obsahu snímku oproti předchozímu, vyhodnotí změnu jako nový snímek prezentace a tento nový snímek prezentace vyexportuje jako samostatný soubor s vyobrazeným slidem z prezentace.

Obrázek 1.2 představuje požadovanou fukci bakalářské práce při detekování videa. První dva snímky jsou shodné, algoritmus je tedy detekoval jako jeden statický frame, který uložil jako samostatný PNG soubor. U snímků 3-6 ale algoritmus detekoval, že po sobě následující snímky ve videu nejsou totožné a neustále se mění. Rozpoznal tedy animaci/video v prezentaci. Tuto animaci/video uložil jako samostatný video soubor.



Obrázek 1.1: Demonstrace fungování práce na statických snímcích



Obrázek 1.2: Demonstrace fungování práce na animaci/videu

Kapitola 2

Analýza a návrh

V této kapitole se věnuji porovnání současných metod pro záznam prezentací v přednáškách a rozpoznávání rozdílných přednáškových slidů ve videu. Dále uvádím přehled různých metod pro nalezení míry rozdílnosti u různých snímků.

2.1 Analýza současných řešení k záznamu interaktivních přednášek

Na internetu lze nalézt mnoho prací zabývajících se zpracováním zaznamenaných přednášek, jejichž výčet naleznete v následujících podkapitolách. Většina z nich vychází z předpokladu, že přednáška je natáčena na jednu kameru, která zabírá jak řečníka, tak prezentované slidy. Jejich cílem je najít okamžiky, kdy došlo ke změně slidu a rozpoznat tento slide díky datům z původního souboru s prezentací (vychází z předpokladu, že mají k dispozici originální PDF nebo PPT soubor s prezentací).

2.1.1 Pomocí QR kódu umístěného na slidu

Autor této myšlenky [1] umisťuje na každý slide prezentace unikátní QR kód, díky kterému lze z videa snadno rozeznat, který slide je zrovna zobrazen. Při postprodukci je zaznamenaný slide s přečteným QR kódem nahrazen původním slidem ze souboru prezentace, ke kterému byl slide s QR kódem předem přiřazen pomocí specializovaného softwaru.

Úspěšnost rozeznání slidu je u této metody velice vysoká, protože QR kódy jsou dobře čitelné i při svém částečném zakrytí, nízkém kontrastu nebo přeexponovaném záběru, který vzniká přepálením obrazu prezentace kvůli dostatečné viditelnosti přednášejícího. Nevýhodou je vkládání těchto QR kódů do každé prezentace ještě před začátkem přednášky. Díky specializovanému programu vytvořenému přesně za tímto účelem nahrávání přednášek je to

otázka pár vteřin, nicméně vložený QR kód může zakrýt část původního obsahu. Přednášející tedy musí předem počítat s volným místem pro QR kód na každé stránce své prezentace. Tato varianta je tedy pro použití v reálném světě a masové rozšíření značně nevhodná.

2.1.2 Čtení obsahu slidů pomocí OCR programu

V této metodě [2] se její autoři zabývají analýzou videa, na kterém je přednášející a přednášená prezentace na projektoru za přednášejícím. Pokud je v záznamu dostatečný kontrast mezi texty a pozadím prezentace lze celé snímky z videa odeslat do programu na rozpoznávání textů z bitmapových obrázků (OCR). Následně se v originálním souboru s prezentací hledají slidy, které mají nejvíce společných slov se slidem, který se právě načetl z videozáznamu.

Výhodou této varianty je zachování výstupních slidů v originální kvalitě (netrpí kompresí video kodeku) a nízká chybovost u čistě textově založených prezentací. Velký ale problém nastává, pokud se na slidech vyskytují obrázky, grafy nebo videa, které program pro rozpoznávání textů nezpracovává. Problémem může být též částečně zakrytý text, nezaostřený obraz nebo málo kontrastní prezentace. Zde může dojít ke špatnému určení slidu nebo k jeho úplnému zahození/nerozpoznání. Metoda tedy není příliš vhodná pro komerční využití.

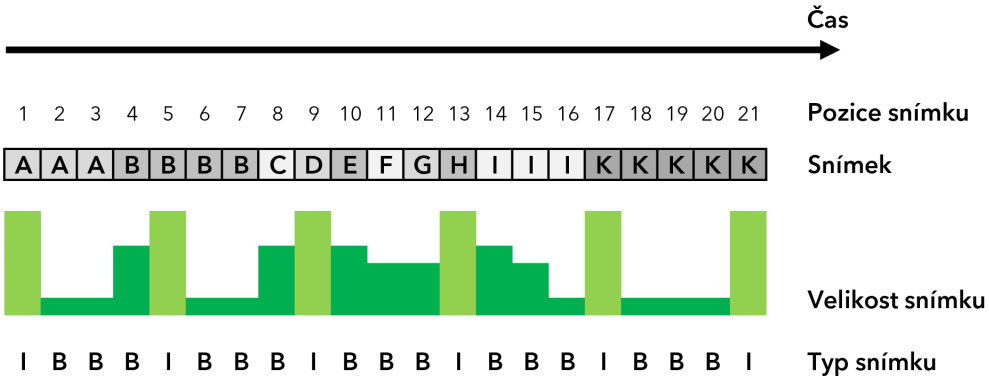
2.1.3 Pomocí rozdílů v datovém toku

Další zajímavou metodou [3] je hledání změn slidů pomocí náhlých změn datového toku ve videu. Podmínkou pro tuto metodu je mít přednášku zaznamenanou ve videu s využitím kodeku, který podporuje variabilní (proměnný) datový tok. Díky tomu, že během přednášek se příliš nemění obsah zaznamenávaného obrazu, jsou kompresní algoritmy schopny snížit datový tok na minimum, protože jedinou změnou v obraze je pohybující se přednášející. Když však dojde ke změně slidu, změní se ve velmi krátkém čase velký počet pixelů a pro dorovnání této změny kodek datový tok prudce zvýší.

Tato metoda je zajímavou možností, která může fungovat velice spolehlivě, pokud má člověk k dispozici pouze čistý záznam obrazovky se slidy. Také pomocí ní lze velice snad rozeznat videa a animace od statických slidů v prezentacích. Problematický je ale předpoklad, že máme video zaznamenané ve formátu s dostatečně variabilním datovým tokem. Žádná nahrávací zařízení (kamera, externí rekordér) nenahrávají s vysoce variabilním datovým tokem kvůli náročné hardwarové implementaci a zanesenému zpoždění. Video je tedy nutné při následném zpracování záznamu převést do vhodného formátu. Díky tomu může kvalita zaznamenaných slidů silně kolísat, protože ne vždy dokáže algoritmus kódující video vhodně zvýšit datový tok při rychlé změně velkého množství pixelů a v obraze tak mohou vznikat kompresní artefakty.

Řešením by mohlo být převádět videa pomocí kodeků podporující více průchodů nebo si z převedeného videa pouze zaznamenat časy změn a reálné slidy a videa vystřihnout z původního záznamu přednášky. Celkový čas potřebný k převedení několikahodinových videí a následného exportování slidů z původního záznamu přednášky ale přesáhne čas, který by člověk strávil manuálním časováním slidů.

U kompresních algoritmů využívajících inter-frame kompresi může dalším problémem být, když změna slidu proběhne přesně v okamžiku klíčového snímku. Tedy snímku, který je komprimován nezávisle na okolních snímcích. Tyto snímky se nazývají I-snímky. Ostatní snímky jsou buď P-snímky, komprimované pouze na základě předcházejících snímků, nebo B-snímky, komprimované i na základě budoucích snímků. Tyto P-snímky a B-snímky mají znatelně menší komprimovanou velikost než I-snímky, pokud se v nich nemění větší množství obsahu. Pokud dojde k výraznému přechodu slidu v P-snímku nebo B-snímku, kompresní algoritmus využije maximální dostupné datové pásmo pro podchycení této skutečnosti. Následkem čehož lze změnu slidu snadno identifikovat. Pokud však změna vyjde na I-snímek, bude velikost snímku stejná, ať dojde ve snímku ke změně oproti předchozímu nebo ne. Video také nesmí být kódované jen pomocí intra-frame komprese, která na rozdíl od inter-frame komprese má všechny snímky jako I-snímky (klíčové) [4].



Obrázek 2.1: Demonstrace problému s klíčovými snímky při rozpoznávání změn pomocí změn v datovém toku

V obrázku 2.1 je demonstrován problém s klíčovými snímky. Můžeme vidět, že každý čtvrtý frame ve videu je klíčový snímek (I-snímek). Je tedy vidět, že velikost klíčových snímků (světle zelená) je podobná, protože kompresní algoritmus kóduje obrázek nezávisle na ostatních. U ostatních snímků je vidět, že pokud se obsah nemění, tak velikost snímku je minimální. Na první pohled můžeme říct, že k nějaké změně dochází ve snímcích na pozících 4 a 8-15. U snímku 15 může být datový tok zvýšen i když se snímek neliší od předchozího

snímku. Důvodem může být, že kompresní algoritmus stále dohání nastalou změnu obrazu, která mohla být velká a navýšení datového toku u jednoho snímku by ji nemuselo pokrýt. Změna, která proběhla u snímku 17 však vyšla na klíčový snímek, nelze ji tedy rozpoznat bez nějakého dalšího porovnání pomocí algoritmů pro porovnávání snímků (viz kapitola 2.2).

Díky nutnosti opětovné komprese je tato metoda značně časově nevýhodná. Je však zajímavou alternativou k jiným přístupům, které pracují s hodnotami jednotlivých pixelů ve videu.

2.1.4 SlidesLive

SlidesLive [5] je český startup založený studenty ČVUT, který k záznamu přednášek používá speciálně vytvořený program, který je dostupný pro Windows a MacOS. Tento program se spustí bez jakékoliv instalace na počítači přednášejícího a v okamžiku vnějšího vstupu (stisknutí prezenteru, tlačítka myši nebo klávesnice, ...) pořídí snímek všech obrazovek připojených k počítači spolu s časovou značkou. Během záznamu nahrává též zvuk, který je snímán mikrofonem počítače.

Po ukončení přednášky se pomocí toho samého programu nahrají zaznamenané snímky prezentace na web `www.slideslive.com`, kde se po nahrání videa se záznamem přednášejícího pořízeného externí kamerou snímky a video samy sesynchronizují podle pořízených audio stop. Výhodou je, že divák si poté může nezávisle na videu posouvat slidy, případně posunout video ke slidu, který ho zrovna zajímá. Celá platforma je zdarma přístupná všem uživatelům internetu.

Tuto metodu ovšem nelze použít, pokud má přednášející počítač s nepodporovaným OS nebo má v počítači zakázáno spouštění programů, které nejsou schválené IT oddělením jeho zaměstnavatele.

2.1.5 Mediasite

Další alternativou je platforma Mediasite od společnosti Sonic Foundry [6]. Jedná se o hardwarovou komponentu, která přijímá odbočený analogový signál z projektoru přes VGA kabel a v reálném čase detekuje snímky ve videu. Platformu lze rozšířit i o další komponenty, díky kterým lze připojit i vstupy z kamer, případně server. Díky nim lze vytvořit automatizované nahrávání, které umožňuje v předem určené časy spustit záznam videa a přednášky a díky uživatelskému rozhraní lze přednášku vzdáleně nastříhat na serveru. Výsledný sestřih lze automaticky nahrát na jednu z podporovaných platform pro přehrávání videí. Systém také podporuje automatické čtení obsahu slidů a analýzu nahraného zvuku, který převádí na text, takže lze v přednáškách snadno vyhledávat obsah. Tato platforma je tedy především určená pro umístění do velkých přednáškových a konferenčních sálů.

Zásadní nevýhodou této platformy je však její vysoká pořizovací cena a kompatibilnost pouze s dalšími komponentami od partnerských společností. Platformu si tedy můžou dovolit jenom velké společnosti. Bohužel kvůli nepřívětivému uživatelskému rozhraní, nutnosti nákupu velkého množství komponent a nemožnosti využít některé komponenty s komponentami od jiných výrobců, leží tyto komponenty ve většině přednáškových sálů často bez využití a záznamy přednášek jsou prováděny pomocí přinesených externích kamer a externích nahrávacích zařízení.

2.2 Algoritmy pro porovnání podobnosti obrazu

V následující části se věnuji výčtu možných metod pro porovnání podobnosti mezi dvěma statickými snímky. Metody jsou řazené od nejzákladnějšího po komplexnější algoritmy.

2.2.1 Odečtení snímků

Tato metoda patří mezi nejzákladnější metody pro porovnání dvou snímků. Při výpočtu dojde k odečtení absolutních RGB hodnot jednotlivých pixelů po barevných kanálech mezi oběma testovanými snímky. Matice hodnot vzniklá odečtem reprezentuje míru rozdílnosti testovaných snímků. Oblasti výsledné matice s vysokými hodnotami, reprezentují velký rozdíl mezi RGB hodnotami u testovaných snímků oblasti s nízkými hodnotami reprezentují téměř shodné oblasti.

Pro výpočet "indexu" rozdílnosti mezi testovanými snímky jsem na výslednou matici vzniklou odečtem aplikoval funkci *threshold*(), která hodnoty menší než stanovený práh nahradí nulou, hodnoty přesahující práh nahradí jedničkou. Na takto vzniklou matici aplikuji součet všech hodnot, který reprezentuje počet pixelů, které jsou mezi testovanými snímky rozdílnější než stanovený práh. Následně stačí tento počet rozdílných pixelů porovnat s celkovým počtem pixelů a zjistíme, kolik procent obrazu se shoduje/neshoduje mezi testovanými snímky.

Protože v prezentacích se často vyskytují barevné nápisy (například červené barvy), tak často dochází ke změně hodnot pouze v jednom barevném kanálu. Výsledný vliv na množství změněné informace by tak nebyl tak výrazný, jak by měl být. Například u nadpisu červené barvy na bílém pozadí dojde ke změně pouze v červeném barevném kanále, pokud má prezentace bílé pozadí, tak ostatní kanály zůstanou beze změny. Celkem tedy dojde ke změně pouze třetiny pixelů v rámci barevných kanálů oproti tomu, kdyby nadpis byl černý. Z toho důvodu pracuje algoritmus použitý v mé bakalářské práci se snímky převedenými do odstínů šedi, kde k tomuto jevu nedochází. Též je díky nižšímu počtu barevných kanálů výpočet rychlejší.

$$\left| \underbrace{\begin{array}{|c|c|c|}\hline 150 & 0 & 255 \\ \hline 150 & 0 & 255 \\ \hline 150 & 0 & 255 \\ \hline \end{array}}_{\text{Hodnoty jednotlivých pixelů ve snímku A}} - \underbrace{\begin{array}{|c|c|c|}\hline 10 & 10 & 10 \\ \hline 255 & 255 & 255 \\ \hline 135 & 135 & 135 \\ \hline \end{array}}_{\text{Hodnoty jednotlivých pixelů ve snímku B}} \right| =$$

$$\begin{array}{|c|c|c|}\hline 140 & 10 & 245 \\ \hline 105 & 255 & 0 \\ \hline 15 & 135 & 120 \\ \hline \end{array} \xrightarrow{\text{threshold()}} \begin{array}{|c|c|c|}\hline 1 & 0 & 1 \\ \hline 1 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline \end{array} \xrightarrow{\text{sum() celé matice}} \text{změnilo se 6 pixelů}$$

V předcházející grafice lze vidět jak probíhá odečítání snímků v reálu pomocí použití funkce *threshold*(), která nahrazuje hodnoty nad mezí (nyní stanovená na 20) za jedničky, hodnoty pod mezí nahrazuje za nuly.

2.2.2 Euklidova vzdálenost dvou snímků

Výpočet této metody je podobný předcházejí metodě. Výsledkem odečítání však není počet rozdílných pixelů, ale vzdálenost obou testovaných snímků od sebe. Tento algoritmus pracuje se snímky jako s plochami v trojrozměrném prostoru, které jsou definované velkým množstvím bodů. Rozměr prostoru je určen počtem barevných kanálů u snímků a jednotlivé body jsou reprezentovány barevnými hodnotami jednotlivých kanálů u pixelů. Pro lepší představu lze použít obrázek 2.2.

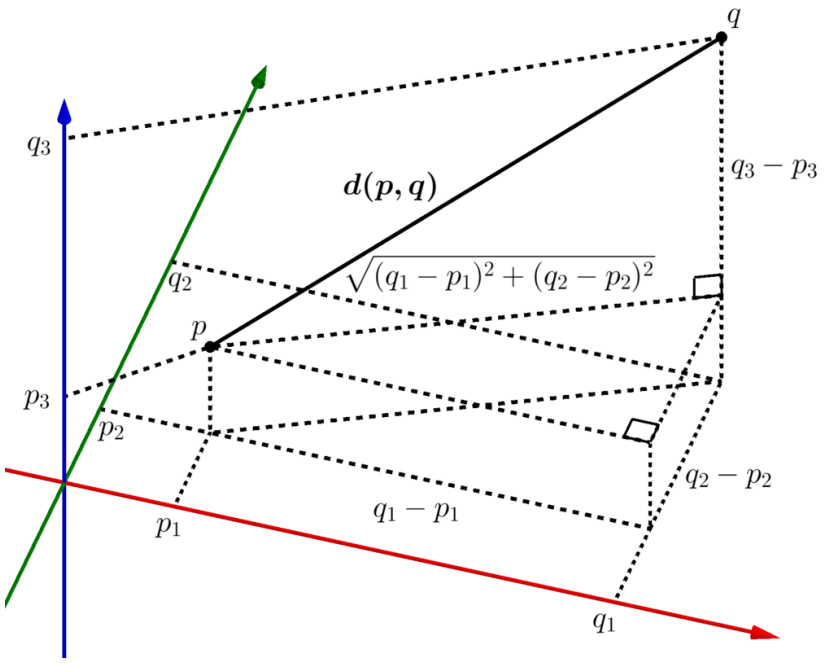
Jednotlivé osy reprezentují barevné kanály a body p a q představují dva rozdílné pixely, respektive dva pixely ze stejných pozic ve dvou různých snímcích. Vzdálenost mezi body p a q se nazývá Euklidovou vzdáleností a lze ji vyjádřit pomocí vzorečku 2.1.

$$d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_n - p_n)^2} \tag{2.1}$$

V praxi porovnávání snímků to znamená od sebe odečíst testované snímky, získanou matici umocnit, sečíst hodnoty všech barevných kanálů pro každý pixel a poté získané součty pro pixely odmocnit. Výsledný součet hodnot ze všech pixelů představuje Euklidovu vzdálenost dvou snímků. Stačí poté porovnat, zda je vzdálenost větší než stanovený práh a pokud ano, jsou snímky považovány za rozdílné.

2.2.3 Peak-signal to noise ratio

Tato metoda vyjadřuje poměr mezi maximální možnou energií signálu a energií šumu. Protože rozdíly signálů můžou být velice rozličné, je $PSNR$ vyjadřováno v logaritmické škále. Základem pro výpočet je střední kvadratická odchylka



Obrázek 2.2: Euklidova vzdálenost

(rozptyl). Ta reprezentuje míru výkyvů daného souboru hodnot od jeho střední hodnoty.

Samotný výpočet v algoritmu začíná absolutním odečtením dvou testovaných snímků. Získaná matice se umocní a sečtou se hodnoty jednotlivých barevných kanálů. V případě, že součet je nulový, dá se říci, že testované snímky jsou naprosto identické. Pokud není, dopočítá se rozptyl (σ^2) pomocí vzorce 2.2.

$$\sigma^{\mathbf{2}} = \frac{1}{c * i * j} \sum (I_1 - I_2)^2, \tag{2.2}$$

ve kterém:

c je rovno počtu barevných kanálů,

i $*$ **j** je rovno celkovému počtu pixelů ve snímku,

$\mathbf{I_1}, \mathbf{I_2}$ jsou matice testovaných snímků.

Samotné $PSNR$ se počítá pomocí vzorečku 2.3.

$$\mathbf{PSNR} = 10 * \log_{10}\left(\frac{MAX^2}{\sigma^2}\right), \tag{2.3}$$

kde MAX reprezentuje maximální hodnotu pixelu, která je v tomto případě 255.

Získaná hodnota $PSNR$ symbolizuje míru podobnosti testovaných snímků. Čím je hodnota vyšší, tím více si jsou obrázky podobné. Ve videích ale dochází k lehké kompresi obrazu, díky které i ten samý slide v dvou různých časech nemá naprosto identické hodnoty pixelů. Proto je třeba počítat s prahem, který dokáže kompresi ignorovat. U videí s průměrnou hodnotou komprese (datový tok FullHD videa ∼20Mbps) se hodnota $PSNR$ u dvou různých snímků se stejným obsahem pohybuje v rozmezí 44-49 dB. Lze tedy říci, že když pro testované snímky vyjde $0 < \mathbf{PSNR} < 43$, tak se jedná o dva různé snímky s rozdílným obsahem jednotlivých pixelů.

2.2.4 Structural similarity

Tato metoda zpracovává obraz více do hloubky a věnuje se samotné struktuře dat uvnitř obrazu, kterou získává pomoci jasového a kontrastního maskování. Jasové maskování je fenomén, který předpokládá, že mírné rozdíly mezi pixely se snadno ztratí ve světlých částech obrazu, zatímco kontrastní maskování předpokládá, že mírné změny se snadno ztrácejí v místech, kde je výrazná textura. $SSIM$ je koeficientem, který vyjadřuje korelaci dvou testovaných snímků a nabývá tedy hodnot -1 až 1, přičemž 1 se rovná naprosto totožným snímkům, -1 se rovná naprosto odlišným snímkům.

Matematicky je výpočet definován jako:

$$\mathbf{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.4}$$

kde:

$\mu_\mathbf{x}$ je průměr hodnot 1. testovaného snímku,

$\mu_\mathbf{y}$ je průměr hodnot 2. testovaného snímku,

$\sigma_\mathbf{x}^\mathbf{2}$ je rozptyl hodnot 1. testovaného snímku,

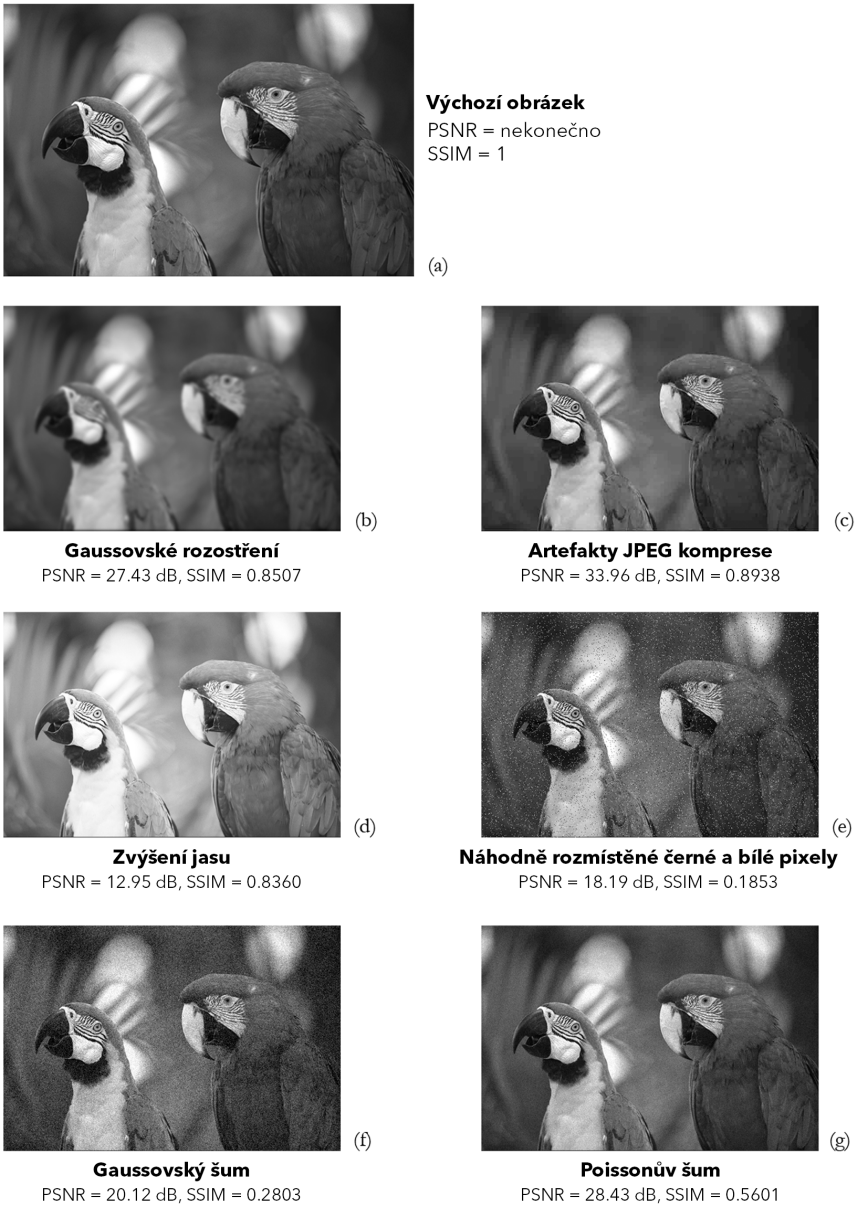
$\sigma_\mathbf{y}^\mathbf{2}$ je rozptyl hodnot 2. testovaného snímku,

$\sigma_\mathbf{xy}$ je kovariance hodnot obou testovaných snímků,

$\mathbf{c_1}$ je hodnota rovna (maximální možná hodnota pixelu $* 0,01)^2 = (255 * 0,01)^2 = 6.5025$,

$\mathbf{c_2}$ je hodnota rovna (maximální možná hodnota pixelu $* 0,03)^2 = (255 * 0,03)^2 = 58.5225$.

Tento přístup je lepší pro rozpoznání odlišnosti běžně vypadajících snímků prezentace, protože se nedívá pouze na samotné hodnoty pixelů, ale respektuje



Obrázek 2.3: Porovnání výsledků algoritmů PSNR a SSIM

i jejich umístění a hodnoty ostatních pixelů ve snímku. Pro příklad můžeme uvést porovnání výchozího obrázku a výchozího obrázku s přidaným kontrastem. Dle základních algoritmů pro porovnání snímků budou snímky odlišné, protože se hodnoty pixelů nebudou rovnat, zatímco $SSIM$ dokáže rozpoznat podobnost obsahu. Když však z původního obrázku polovinu zakryjeme, tak

podle základních algoritmů budou snímky z poloviny stejné, zatímco SSIM vyhodnotí, že se výrazně liší.

Porovnání úspěšnosti algoritmů PSNR a SSIM je znázorněno v obrázku 2.3 [7, 8]. Pro připomenutí, čím vyšší hodnota, tím jsou podle algoritmů obrázky podobnější. Jak lze vypozorovat, na některé typy úpravy obrazu jsou některé algoritmy méně citlivé, například algoritmus $PSNR$ si dobře poradí s Gaussovským nebo Poissonovým šumem, ale zvýšení jasu mu dělá problémy, na rozdíl od algoritmu $SSIM$, který na zvýšení jasu nereaguje jako na úplně odlišný obrázek.

Celkově lze ale říci, že pokud algoritmu $SSIM$, který pracuje se snímkem jako s celkem, předložíme z velké části stejné snímky jen s několika výrazně upravenými náhodně vybranými pixely, tak začíná obrázky detekovat jako odlišné; na rozdíl od podobné změny aplikované na všechny pixely (například zvýšení jasu). Oproti tomu algoritmus $PSNR$, který pracuje s jednotlivými pixely, je necitlivý na změny pár pixelů, nicméně pokud změníme všechny pixely, detekuje snímky jako odlišné.

2.2.5 Hledání hran pomocí Sobelova operátoru spojené s odečítáním snímků

Texty v prezentacích jsou převážně kontrastní vůči pozadí, lze tedy využít metody pro hledání hran v obraze a výsledné matice s výstupem prezentujícím výskyt hran porovnat. Výhodou této metody je, že je invariantní vůči šumu s nízkou absolutní intenzitou a tedy částečně i artefaktům vzniklým při kompresi videa. Porovnávacímu algoritmu dokáže předložit čisté kontrastní matice.

K hledání hran jsem se rozhodl využít Sobelův operátor, který je prezentován pomocí dvou matic:

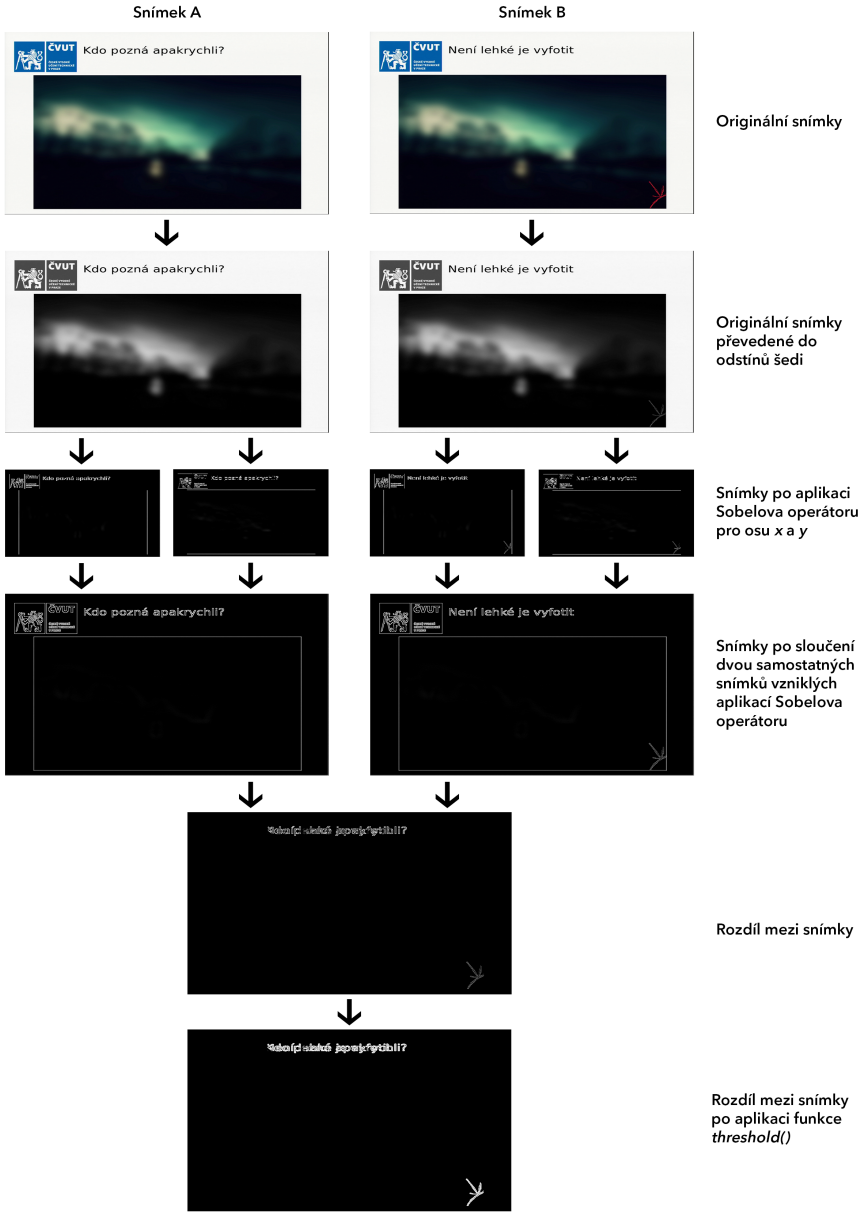
$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{I} \qquad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{I},$$

kde I reprezentuje vstupní snímek. Matice $\mathbf{G}_x$ detekuje hrany v ose x, matice $\mathbf{G}_y$ detekuje hrany v ose y. Výsledná matice se získá vložením matic filtrovaných podle osy x a y do vzorce 2.5.

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2} \tag{2.5}$$

Výsledné získané matice s hranami z obou testovaných snímků se následně porovnají pomocí algoritmu odečítání snímků s thresholdem, který je popsán výše.

Celý proces porovnání dvou statických snímků je znázorněn v obrázku 2.4. Jak lze vidět, algoritmus správně detekoval rozdílný nadpis a přidání červené čáry pro vyznačení pravého dolního rohu, ve kterém se vyskytují Apakrychle.



Obrázek 2.4: Ukázka procesu porovnávání snímků pomocí algoritmu na hledání hran se Sobelovým operátorem

Pro představu v tomto případě podle algortimu došlo ke změně 23 476 pixelů z celkového počtu 2 073 600 pixelů, tedy $\sim 1.13\%$ všech pixelů ve snímku.

2.2.6 Perceptual hashing

Jedná se o metodu, během které se pro různé typy multimédií získává unikátní otisk (hash), který je založený na různých vlastnostech média [9]. V případě matic s hodnotami jednotlivých pixelů se pracuje s obsahem matice. Dle způsobu hashování lze získat různě dlouhý hash, u kterého platí, že čím delší je, tím více informací obsahuje. Stejné obrázky budou produkovat stejné hashe, proto díky podobnosti hashe dvou obrázků lze předpokládat i jejich podobnost v obsahu.

V rámci tématu mé bakalářské práce je tato metoda nevhodná pro porovnání dvou různých snímků prezentace, protože i malé artefakty ve snímcích se stejným slidem jenom z jiného okamžiku prezentace generují odlišný hash. Proto tento algoritmus není implementován v praktické části této práce. Nicméně v oblastech vyhledávání podobných obrázků v rámci velkých obrazových sad je tato metoda velice často používána. Je totiž mnohem rychlejší porovnávat tisíce různých hashů, než tisíce velkých obrázků. V případě, že se hashe dvou obrázků podobají, aplikuje se lepší porovnávací metoda, která je sice časově více náročná, ale dokáže podobnost obrázků upřesnit.

2.3 Porovnání algoritmů

V této sekci se věnuji porovnání jednotlivých představených algoritmů a vyzdvihnutí jejich výhod a nevýhod na různé druhy přednášek. Testování probíhá s metodami, které jsem implementoval v rámci své bakalářské práce a vytvářel jsem je v co nejoptimálnějším řešení. Výsledný skript s naměřenými časy je pouštěn na mém notebooku Asus GL502VS (Intel Core i7-6700HQ, SSD, Windows 10). Na jiných zařízení se můžou časy běhu programu lišit.

Jako testovací prezentaci jsem použil záznam přednášky Apakrychle aneb fenomén dnešní doby [10], jejíž záznam jsem pořídil během Informatického večera na FIT ČVUT 21. února 2017. Volba na tuto přednášku nepadla jenom z důvodu, že je přednášena lidmi z FITu, ale především kvůli jejímu obsahu. Přednáška obsahuje dvě videa, je v ní dostatek slidů, které se liší jak běžným způsobem, tak i například o jeden přidaný znak. V přednášce jsou přidávány i nadpisy různých barevných kombinací, tenké čáry a obrázky. Obsahuje i oblast, kde přednášející vypne svoji prezentaci a hledá prezentaci svého kolegy, je zde tedy i část s pohybem myši. Tím přednáška obsahuje většinu případů, které mohou během přednášení nastat.

Pro testování mezních případů jsem nasimuloval prezentaci, která obsahuje nejčastější mezní případy:

- přechod mezi bílou a černou obrazovkou pro testování velkého kontrastu,
- přidání světle šedého textu na bílé pozadí pro testování malého kontrastu,

- přidání malé kontrastní tečky do snímku,
- dlouhé prolínání přidávaného textu pro testování dlouhé animace s malým počtem rozdílných bodů,
- přechod mezi dvěma stejnými snímky s efektem posunutí, pro testování stejného obsahu ve snímku ale na jiné pozici,
- přechod mezi dvěma snímky, z nichž jeden má pravou část černou a levou část bílou a druhý je k němu inverzní, pro testování stejného obsahu snímku jenom na jiném umístění,
- programovací část, ve které je psán kód, pro testování rozpoznání videa.

Tuto prezentaci s krajními případy jsem vyexportoval jak s vysokým datovým tokem [11] pro samotné testování mezních případů, tak i s velmi malým datovým tokem [12], kde je obraz hodně degradován kompresí video kodeku.

Všechny testované soubory jsou k dispozici ke stažení na Google Drive. Přednáška o Apakrychlích [13], video s mezními hodnotami s vysokým datovým tokem [14] a video s mezními hodnotami s nízkým datovým tokem [15].

2.3.1 Odečtení snímků

Díky své jednoduché implementaci patří tato metoda k nejrychlejším ze všech testovaných metod (viz tabulka 2.1). V algoritmu rozhodují dva parametry. Jedním z nich je práh, který definuje, jak moc rozdílná musí být hodnota po odečtení obou testovaných snímků, aby byl pixel považován za odlišný, a druhá definuje kolik procent pixelů z celkového počtu pixelů ve snímku se musí lišit, aby byl snímek považován za rozdílný.

Díky tomu, že texty v prezentacích jsou většinou kontrastní, osvědčil se práh 20 jednotek. Tj. kdykoliv je mezi těmi samými pixely na stejných místech rozdíl vyšší než 20 jednotek RGB, algoritmus pixely vyhodnotí jako rozdílné. Problém nastává u snímků s videem, kde se obsah mění postupně a algoritmu trvá delší dobu, než se hodnota dostane za prahovou hodnotu. Tato doba bohužel může trvat tak dlouho, že algoritmus považuje snímek za statický a vzniká tak false negative.

Na základě delšího experimentování vyšel nejlépe i pro zachycení těch nejmenších změn počet rozdílných pixelů v obraze 0.01% z celkového počtu pixelů. Ačkoliv se tato hodnota může zdát málá, tak například přidání znaků $+c$ v čase 16:27 v přednášce o Apakrychlích [10] zabírá cca 250 pixelů, což je z celkového počtu $1920 * 1080$ pixelů přibližně 0.012%.

Celkově lze říci, že algoritmus je díky své jednoduchosti velice rychlý a jeho výsledky jsou překvapivě kvalitní. Je to způsobené především metodou *threshold*(), která umožní ignorovat velké množství šumu, který se v obraze nachází, a výpočty se poté soustředí pouze na velké změny. V testovaném videu se záznamem přednášky o Apakrychlích [10] se mu podařilo detekovat

všechny slidy a došlo pouze k jedné chybě při vyhodnocení videa v prezentaci. Tady byl rozdíl mezi snímky ve videu velice malý a proto algoritmus toto místo vyhodnotil jako sekvenci objektů *video - statický snímek - video - statický snímek - video*, přitom správně by to mělo být jedno dlouhé video. I tak se algoritmus ukázal jako velice výkonný a spolehlivý pro analýzu běžných přednášek.

U videa s mezními případy [11] si algoritmus též vedl dobře. Podařilo se mu rozeznat všechny slidy bez false positive u verze s vyšším datovým tokem. Problém nastal u dlouhého přechodu, který u verze s vyšším datovým tokem interpretoval jako sekvenci statických snímků. Programování interpretoval jako sérii krátkých videí, nicméně tento neduh by se dal opravit nastavením delšího času pro identifikaci doby trvání videa (viz kapitola 3.2.2).

S videem s mezními případy s nízkým datovým tokem si příliš dobře nevedl. Část se slidy detekoval díky velkým kompresním artefaktům jako jedno dlouhé video. Část s programováním již ale detekoval jako jedno dlouhé video, zde si tedy poradil správně, ačkoliv to bylo způsobeno kompresí videa.

2.3.2 Euklidova vzdálenost dvou snímků

Výpočetní náročnost této metody je o mnoho vyšší než u jiných algoritmů především kvůli velkému množství matematických operací při výpočtech. Z hlediska nalezení běžně rozdílných slidů nemá problém a poradí si i s animací a barevnými nadpisy. Potíž nastává u drobných změn jako například přidání symbolu $+c$ v čase 16:27 v přednášce o Apakrychlích [10].

Tato změna je tak malá, že se ztratí v šumu způsobeném kompresí. Euklidova vzdálenost dvou stejných snímků při běžné úrovni komprese u FullHD videa je okolo 3 800 000 +/- 200 000 jednotek. Přidaní znaků $+c$ na slidu v čase 16:27 přednášky o Apakrychlích zabírá přibližně 250 pixelů, které se prolnou z bílé na černou. Když to přepočítáme na Euklidovu vzdálenost, tak se jedná o $\sqrt{(255-0)^2+(255-0)^2+(255-0)^2} * 250 = 191250$ jednotek, což vzhledem k odchylce šumu nelze spolehlivě zachytit. Při snížení prahu pro detekci změny snímku dojde k falešné detekci změn snímků (false positive), které jsou způsobené kompresí videa.

I tak se podařilo při nejvhodnějším nastavení hraničního prahu odchytit téměř všechny změny snímků prezentace, kromě snímků v čase v 15:57 (přidání slovíčka *Žádná!*) a 16:27 (přidání slovíčka $+c$). Ve videu nastaly i dva případy false positive, kdy algoritmus rozhodl, že přepínání přednášky v čase 18:29 rozdělí na dvě videa se statickým snímkem v 18:33.

Při testu s prezentací s mezními případy [11] algoritmus oproti ostatním hodně zaostává. Při vyšším ani nižším datovém toku nedokázal detekovat malé změny slidu jako přidání malé tečky a šedého textu na bílé pozadí. U verze s vyšším datovým tokem detekoval dlouhé prolínání jako statický snímek a programování jako sérii čtyř statických snímků a jednoho vteřinového videa. Při nižším datovém toku detekoval velké množství false positive

snímků včetně animací na statických snímcích. Programování detekoval jako sérii statických snímků.

Tento algoritmus lze tedy považovat za celkem spolehlivý, pokud máme přednášku, která obsahuje velké rozdíly mezi slidy a neobsahuje příliš videa.

2.3.3 Peak-signal to noise ratio

Metoda se osvědčila jako velice rychlá na výpočet a kvalitu rozpoznání malých změn mezi snímky. Díky soustředění se na velké rozdíly netrpí na false positive detekce u snímků s velkým šumem. Též je schopna detekovat i malé rozdíly v rámci videa. Pro příklad mohu uvést, že například algoritmus odečítání snímků nepozná rozdíl mezi tím, jestli změním v obraze jeden pixel o 10 jednotek nebo všechny pixely o 10 jednotek RGB. Z jeho pohledu se jedná o dva stále stejné snímky, protože změna o 10 jednotek se nachází pod prahem pro detekci šumu. $PSNR$ je schopno rozeznat mezi těmito snímky velký rozdíl.

V testech na videu s přednáškou o Apakrychlích [10] se jí úspěšně povedlo detekovat všechny slidy a videa včetně těch s malými změnami. Také se díky jednoduchosti svého výpočtu umístila mezi nejrychlejšími algoritmy.

V testu s prezentací s mezními případy [11] s vyšším datovým tokem se jí podařilo detekovat všechny slidy, avšak poslední část s programováním a pomalým přechodem s načtením jednoho titulku kvůli malým změnám algoritmus vyhodnocoval jako statické snímky měnící se každou vteřinu. Tento problém by se ale pravděpodobně dal vyřešit pomocí snížení prahu.

U verze s nízkým datovým tokem [12] došlo k několika chybně identifikovaným snímkům (false positive) způsobenými velkou kompresí videa. Úvodní část se slidy byla celá vyhodnocena jako video a závěrečné programování bylo z převážné části vyhodnoceno jako série statických snímků, protože díky kompresi došlo k velké ztrátě detailů ve videu.

2.3.4 Structural similarity

Ačkoliv dle odborné literatury má tato metoda patřit k jedné z nejlepších pro hledání rozdílů mezi snímky, pro detekování snímků v prezentacích se moc neosvědčila. Všeobecně je u ní problém s nalezením vhodného prahu. Když je práh nastavený moc nízko, některé snímky splynou ve více a nejsou detekovány malé rozdíly a videa jsou rozdělena na více částí, protože algoritmus uprostřed videa vyhodnotil, že video je statickým snímkem, tj. bylo v něm málo změn. Pokud se práh začne zvedat, tak se videa začnou spojovat v celistvé celky, ale díky kompresi videa začnou vznikat false positive případy, kdy se statické snímky detekují jako video.

Ve videu o Apakrychlích [10] se algoritmu podařilo detekovat všechny snímky krom jednoho černého, který předchází videu a algoritmus jej zahrnul

do následného videa. Problém nastal u videí, kde algoritmus videa rozdělil do mnoha kratších videí a statických snímků.

V prezentaci s mezními případy s vysokým datovým tokem [11] se podařilo detekovat všechny slidy, ale video o programování se rozdělilo na velké množství statických snímků a jedno krátké video. Problémem je malá změna v obraze, díky které se změny nevejdou do mezí stanových prahem. U verze prezentace s nízkým datovým tokem [12] se algoritmus zachoval stejně jako ostatní, kdy první část se statickými snímky detekoval jako souvislé video a programování rozdělil mezi mnoho statických snímků.

2.3.5 Hledání hran pomocí Sobelova operátoru spojené s odečítáním snímků

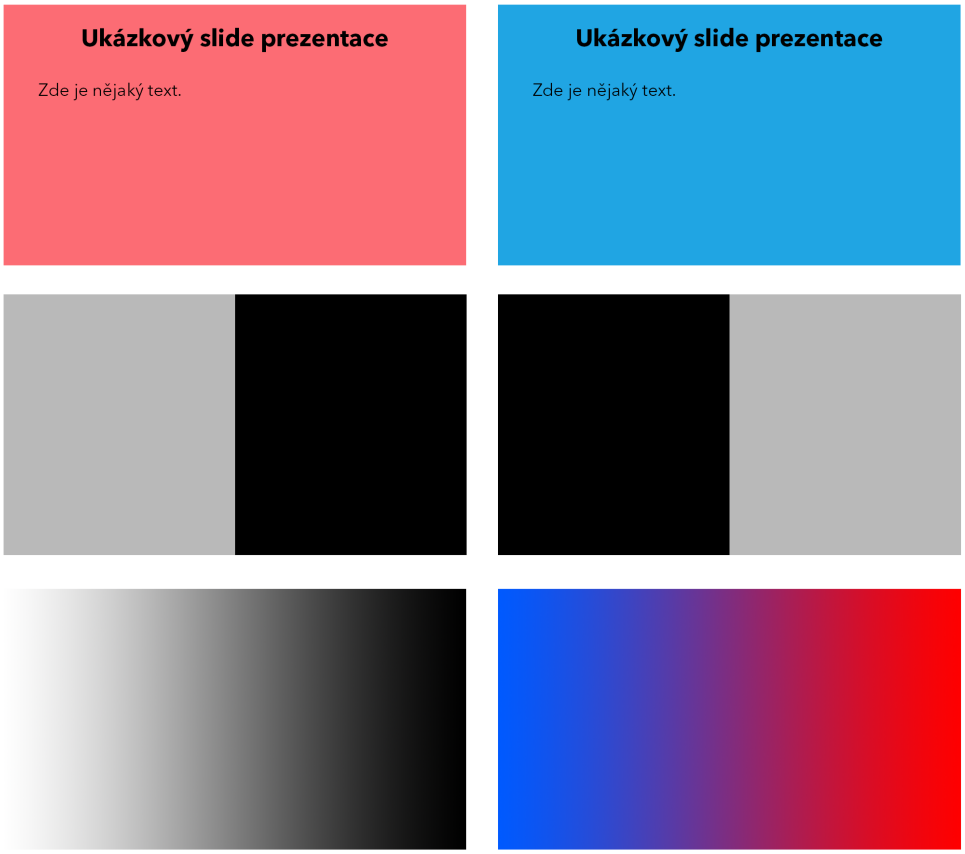
V této metodě se počítá především s rozdíly mezi hranami v obou testovaných snímcích. Díky tomu, že ve většině přednášek jsou texty kontrastní vůči pozadí, lze práh pro rozhodnutí o rozdílnosti snímků nastavit na pouhých 0.015% rozdílných pixelů z celkového počtu pixelů ve snímku. Menší problém nastává u snímků, u kterých se změní pouze barvy, ale kontrastní místa zůstávají. Příkladem může být přechod mezí kompletně bílým a černým snímkem. Problém také trochu nastává u videí, ve kterých nemusí být mezi jednotlivými snímky dostatečně rozdílné oblasti s kontrastními hranami.

Ve videu o Apakrychlích [10] se podařilo korektně detekovat všechny snímky vyjma videa s hypnožábou v čase 10:34, kterou algoritmus rozdělil na dvě samostatná videa a jeden statický snímek.

Prezentace s mezními případy [11] ukázala velké slabiny tohoto algoritmu. Snímky, ve kterých se střídá jednobarevný snímek černé a bílé barvy, nebyly vůbec detekovány, stejně jako snímek, kde je jedna polovina obrazu černá, druhá polovina bílá, a následující snímek je tady k tomu inverzní. Algoritmus bohužel detekuje pouze rozdíly mezi hranami, ne rozdíly v barevném rozložení. V programovací části se ale algoritmus ukázal jako nejlepší ze všech testovaných. Celé programování rozdělil mezi 2 videa a 1 statický snímek.

Ve variantě s nižším datovým tokem [12] došlo k prolnutí vlastností tohoto porovnávacího algoritmus a chyb, které zde učinily i ostatní testovací algoritmy. Nepodařilo se detekovat jednobarevné slidy a úvodní část se slidy se detekovala jako jedno dlouhé video. Programovací část se ale podařilo detekovat jako jedno dlouhé video, což se žádnému jinému algoritmus krom odečítání snímků nepodařilo.

V obrázku 2.5 lze vidět ukázky snímků, které tento algoritmus vyhodnotí jako shodné snímky, protože se v nich shodují pozice kontrastních hran, respektive v nich žádné kontrastní hrany nedetekuje.



Obrázek 2.5: Ukázka mezních případů Sobelova algoritmu

2.3.6 Shrnutí

Každý testovací algoritmus má své výhody a nevýhody. Pro přehlednost přikládám tabulku 2.1 s výsledky testování na videu o Apakrychlích [10], tabulku 2.2 s výsledky testování na testovacím videu s mezními případy s vysokým datovým tokem [11] a tabulku 2.3 s výsledky testování na testovacím videu s mezními případy s nízkým datovým tokem [12]. V prvním sloupci je uveden název algoritmu využitého pro detekci, druhý uvádí jaký počet statických snímků a videí našel a třetí za jaký čas. Protože celkový čas je silně ovlivněn ukládáním videa, které je nejvíce časově náročnou operací celého algoritmu, je pro menší zkreslení primárně uveden čas bez ukládání videí, v závorce je poté čas i s ukládáním videí. Důvod je prostý, algoritmy, které špatně detekují video jako sérii statických slidů, by v celkovém čase dopadly lépe než ty, které jej detekují správně. Poslední dva sloupce uvádějí počet false positive a false negative, tedy počet rozdílných snímků, které algoritmus detekoval na-

Tabulka 2.1: Porovnání časů a úspěšností všech algoritmů pro video o Apakrychlích

Algoritmus	Počet detekovaných slidů (snímků / videí)	Čas běhu (čas běhu s exportem videí)	False positive	False negative
Odečtení snímků	107/5	2:18 (4:48)	4	0
Euklidova vzdálenost	104/4	4:48 (7:11)	2	2
PSNR	105/3	2:51 (5:24)	0	0
SSIM	108/7	3:20 (5:40)	8	1
Detekce hran	106/4	4:26 (6:58)	2	0

Tabulka 2.2: Porovnání časů a úspěšností všech algoritmů pro video s mezními případy s vysokým datovým tokem

Algoritmus	Počet detekovaných slidů (snímků / videí)	Čas běhu (čas běhu s exportem videí)	False positive	False negative
Odečtení snímků	19/3	0:54 (2:14)	10	0
Euklidova vzdálenost	14/2	0:31 (0:47)	5	2
PSNR	22/5	0:49 (1:37)	13	0
SSIM	24/2	0:58 (1:49)	12	0
Detekce hran	11/3	1:04 (2:42)	5	5

víc, a počet snímků, které algoritmus vůbec nedetekoval. Je tedy lepší mít více false positive a mít nalezené některé slidy duplikovaně, než mít více false negative, kdy algoritmus snímek nedetekoval a ve finálním výstupu chybí.

U videa o Apakrychlích je správný počet 105 statických snímků a 3 videa. U videa s mezními případy je správně 11 statických snímků a 4 video soubory s animacemi a programováním.

Jak lze z tabulek i popisků vyčíst, nelze jednoznačně určit, který algoritmus je pro detekování snímků v prezentaci nejlepší. Záleží na obsahu prezentace a taky na kvalitě záznamu. Lze ale říci, že pro běžné prezentace, které obsahují převážně statické snímky, se jeví jako nejvíce vhodný algoritmus $PSNR$, který

Tabulka 2.3: Porovnání časů a úspěšností všech algoritmů pro video s mezními případy s nízkým datovým tokem

Algoritmus	Počet detekovaných slidů (snímků / videí)	Čas běhu (čas běhu s exportem videí)	False positive	False negative
Odečtení snímků	12/2	0:23 (1:14)	3	4
Euklidova vzdálenost	34/4	0:52 (1:03)	26	2
PSNR	31/5	0:36 (1:03)	24	2
SSIM	24/4	0:40 (1:15)	16	3
Detekce hran	5/2	0:38 (1:31)	2	8

zaujal nejen spolehlivostí při hledání snímků, ale i časem za jaký dokázal video analyzovat. Pokud by však měl záznam obsahovat velké množství videí nebo programování, doporučil bych použít algoritmus pro detekci hran pomocí Sobelova operátoru. Algoritmus sice patří na čas k těm nejpomalejším, avšak v oblasti rozeznávání videí a záznamu obrazovky s programováním převedl nejlepší výkony.

Kapitola 3

Realizace

V této kapitole se věnuji implementaci zvoleného přístupu a použitým technologiím v praktické části mé bakalářské práce.

3.1 Použité technologie

Celý program je napsán v jazyce C++ a je testován a laděn na operačním systému Windows.

3.1.1 OpenCV

Ve své bakalářské práci používám pro práci s videem knihovnu OpenCV (Open Source Computer Vision Library). Knihovna je volně šiřitelná pod BSD licencí a je volně použitelná pro akademické i komerční účely. Její interní funkce a metody jsou vysoce optimalizovány a jsou přizpůsobeny pro výpočty ve více vláknech. Je vyvíjena komunitou více než 47 tisíc lidí a je široce používaná v oblasti práce s videem [16].

V své práci používám verzi OpenCV 3.4.1, která vyšla 27. ledna 2018 [17]. Z knihovny používám funkce pro načítání a ukládání video souborů a obrázků a funkce pro základní matematické operace s maticemi. Přesněji knihovny *opencv_world341.dll* a *opencv_ffmpeg341_64.dll*.

3.1.2 OpenH264

OpenH264 je knihovna pro práci s H.264 kodekem implementovaná od společnosti Cisco. Knihovna je volně šiřitelná pod licencí BSD. Původně byla vytvořena pro možnost přehrávání videí s kodekem H.264 ve webových prohlížečích a také pro možnost pracovat s kodekem H.264 bez nutnosti platit za licenci MPEG LA [18].

V mé práci je knihovna využívána k ukládání výstupního videa/animací v kodeku AVC1/H.264.

3.2 Algoritmus pro zpracování výsledků z porovnávacích metod

V následující sekci je kód bakalářské práce rozdělen do více logických podsekcí, které se zabývají jednotlivými celky ve zdrojovém kódu. Program je koncipován jako třída, kterou uživatel může využít do svého vlastního programu.

Celý program je rozdělen do 3 hlavních celků:

třída presentationAnalyzer hlavní třída, která obsahuje všechny funkce potřebné pro výpočty,

struktura valuesSet struktura, která obsahuje všechny pevně nastavené hodnoty pro výpočty a porovnávání,

třída videoSettings třída, která je určená pro nastavení parametrů analýzy.

Popis jednotlivých metod je k dispozici v dokumentaci včetně konkrétních názvů a vstupních a výstupních hodnot. Následující podsekce se zabývají funkcionalitou uvnitř jednotlivých metod a popisem algoritmu pro analýzu.

3.2.1 Základní algoritmus pro zpracování videa

Níže se nachází popis celkového procesu pro zpracování videa se záznamem přednáškových slidů od jeho otevření po finální uložení výstupních souborů. V rámci zpřehlednění následujících bodů jsou jména následujících proměnných nahrazeny jejich překladem:

BASIC_CHECK_EVERY_X_SECONDS interval pro kontrolu nových snímků,

BASIC_CHECK_EVERY_X_FRAMES_TO_FINISH_SLIDE interval pro kontrolu dokončení přechodu,

BASIC_SECONDS_TO_FINISH_SLIDE čas k dokončení přechodu.

1. Program otevře video soubor, který mu uživatel na vstupu přiřadí pro analýzu. Zkontroluje, zda je soubor v pořádku, a načte jeho základní parametry (délku, snímkovou frekvenci, rozlišení, ...). Pokud video není validní, program se v tomto kroku ukončí.

2. Vytvoří se složka pro výstupní soubory a soubor XML pro ukládání časových značek. Pokud se vytváření z jakéhokoliv důvodu nezdaří, program se v tomto kroku ukončí.

3. Program načte první snímek videa a uloží jej jako první slide prezentace a také jako *výchozí snímek*. *Výchozí snímek* slouží k porovnávání se snímkem na aktuální pozici při procházení videa. Program také nastaví *aktuální pozici* ve videu na tento první snímek.

4. Algoritmus přičte k *aktuální pozici časový interval pro kontrolu nových snímků* a přečte snímek na dané pozici. Dle zvoleného porovnávacího algoritmu porovná *výchozí snímek* a snímek na *aktuální pozici*. Pokud:

   - porovnávací algoritmus vyhodnotí, že se snímky shodují, algoritmus opakuje krok č. 4 znovu,
   - se snímky neshodují, algoritmus přejde do dalšího kroku.

5. Algoritmus od *aktuální pozice* odečte *časový interval pro kontrolu nových snímků* a získanou pozici považuje za *počáteční pozici*. Nyní algoritmus ví, že ke změně slidu došlo někde mezi těmito pozicemi. Pomocí binárního půlení a rekurze najde přesnou pozici snímku, který se jako první liší od *výchozího snímku* a považuje jej za okamžik, kdy došlo ke změně snímku prezentace nebo začátku animace. Algoritmus vytvoří proměnnou s *pozicí poslední změny* a uloží do ní čas, kdy našel první odlišný snímek od *výchozího snímku*. Také nastaví *aktuální pozici* shodně s *pozicí poslední změny*.

6. Algoritmus přičte k *aktuální pozici interval pro kontrolu dokončení přechodu* a načte snímek na této pozici. Ten porovná se snímkem na *pozici poslední změny*. Pokud:

   - se snímky liší, znamená to, že mezi snímky probíhá nějaká animace/přechod/video. Algoritmus tedy aktualizuje *pozici poslední změny* shodně s *aktuální pozicí* a znovu pustí krok č. 6,
   - se snímky shodují, tak zjistí, jak dlouhý čas uplynul od *pozice poslední změny* k *aktuální pozici*. Pokud je čas:
     - menší než *čas k dokončení přechodu*, znovu proběhne krok č. 6,
     - větší než *čas k dokončení přechodu*, algoritmus rozpozná, že animace/přechod/video skončilo a přechází do dalšího kroku.

7. Algoritmus zjistí, jak velký čas uplynul mezi *počáteční pozicí* a *pozice poslední změny*. Zjistí tím, jak dlouho trval přechod/animace/video. Pokud:

   - je tato doba menší než *čas k dokončení přechodu*, považuje přechod za statický. Vezme tedy snímek na *pozici poslední změny* a uloží jej jako statický snímek,
   - je tato doba větší než *čas k dokončení přechodu*, považuje přechod za video/animaci. Uloží tedy celou dobu mezi *počáteční pozicí* a *pozici poslední změny* jako jeden video soubor.

8. Algoritmus uloží jako *výchozí snímek* snímek, který je na *pozici poslední změny*. Pak se vrátí do kroku č. 4.

Pokud kdekoliv během kroku č. 4 dojde ke konci videu, program se ukončí. Pokud dojde ke konci videa v kroku č. 5 nebo č. 6, tak přejde na krok č. 7 a následně se program ukončí.

3.2.2 Definované hodnoty použité pro analýzu

Následující odstavce se věnují hodnotám uvnitř třídy valuesSet. Jejich výpis je k dispozici v přiložené tabulce 3.1. Všechny tyto hodnoty byly získány na základě pozorování a testování. Měřením se podařilo zjistit, že časově nejvíce náročné operace jsou čtení a ukládání obrázků z/do video souboru. Například načtení 1000 snímků z video souboru trvá 34 vteřin, zatímco uložení 1000 snímků trvá 57 vteřin. Samotné čtení a ukládání může tedy znamenat až 80% času běhu samotného programu. Mým cílem bylo tedy tyto operace maximálně omezit.

Proměnná BASIC_CHECK_EVERY_X_SECONDS je stanovena na 4 vteřiny, což znamená, že při běžném procházení algoritmus čte nový snímek z videa až po přetočení čtyř vteřin záznamu a porovnává jej s výchozím snímkem, kde je uložen poslední slide prezentace. Když je interval kratší, dochází ke zbytečně častému čtení snímků se zachováním stejné úspěšnosti detekce slidů. Pří delším intervalu naopak dochází k nutnosti procházet velké množství snímků mezi dvěma kroky k nalezení začátku změny slidu prezentace, viz krok č. 5 v algoritmu 3.2.1.

BASIC_CHECK_EVERY_X_FRAMES_TO_FINISH_SLIDE definuje, jak často bude algoritmus načítat snímek ze souboru, aby mohl porovnat, jestli se už přechod ustálil nebo animace/přechod/video stále trvá. Jelikož kontrola každého snímku by nebyla příliš časově efektivní, po několika testech se ukázala jako vhodná volba kontrolovat každé 3 snímky.

V proměnné BASIC_SECONDS_TO_FINISH_SLIDE je definováno, jak dlouho musí být obraz statický (beze změn), aby mohl algoritmus předchozí video / snímek vyhodnotit za ukončený. Pokud je interval kratší, dochází k rozdělení videí, protože ve videu může nastat situace, kdy se obraz delší dobu nemění, například přechody přes černou barvu. Pokud by byl interval delší, mohlo by docházet ke splynutí snímků, kdy přednášející rychle proklikává snímky a tím by se snímky detekovaly jako jedno video místo mnoha statických snímků.

Další hodnoty jsou získané opakovaným spouštěním algoritmů a testováním hraničních mezí, které jsou nad úrovní šumu způsobeného komprimací videa.

Všechny proměnné si může uživatel dle potřeby změnit.

3.2.3 Čtení a ukládání dat

Pro práci s multimediálními soubory je použita knihovna OpenCV. Před samotným uložením je vytvořena složka pro ukládání. Statické snímky jsou uklá-

Tabulka 3.1: Hodnoty třídy valuesSet

Název proměnné	Hodnota	Jednotky
BASIC_CHECK_EVERY_X_SECONDS	4	sekundy
BASIC_CHECK_EVERY_X_FRAMES_TO_FINISH_SLIDE	3	snímky
BASIC_SECONDS_TO_FINISH_SLIDE	0.8	sekundy
THRESHOLD_PSNR	43.5	dB
THRESHOLD_DIFF_DIFFERENCE_BETWEEN_PIXELS	20	RGB hodnoty (0-255)
THRESHOLD_DIFF	0.1	procenta
THRESHOLD_SSIM	0.9992	*bez jednotek*
THRESHOLD_SOBEL_DIFFERENCE_BETWEEN_PIXELS	25	RGB hodnoty (0-255)
THRESHOLD_SOBEL	0.015	procenta
THRESHOLD_EUCLID	1.7	RGB hodnoty (0-255)

dány jako PNG obrázky, videa a animace jsou ukládány jako MP4 soubory s video kodekem AVC1/H.264. Samotné soubory jsou uložené pomocí názvů ve tvaru *pořadové číslo snímku-čas změny snímku od počátku videa ve tvaru (HH-MM-SS-mmm)*. Takže soubor *0096-00-46-50-096.png* prezentuje 96. nalezený slide v pořadí od začátku video souboru s prezentací a slide se poprvé v obraze objevil v čase 46 minut, 50 vteřin a 96 milisekund.

Součástí je i XML soubor *timestamps.xml*, který obsahuje časové značky se změnami jednotlivých slidů. Jeho struktura je zobrazena v obrázku 3.2.3.

`slides` ..... reprezentuje celou prezentaci
- `slide` ..... samostatný slide prezentace
  - `filename` ..... název souboru se slidem/animací včetně přípony
  - `timeInVideoMillis` .... počet milisekund od počátku video souboru s prezentací, kdy došlo k detekci slidu/animace
  - `timeInVideoReal` ..... lidsky čitelný čas, kdy došlo k detekci slidu/animace od počátku video souboru s prezentací
  - `text` ..... typ média - *static* pro statické snímky, *video* pro animace a videa

Obrázek 3.1: Struktura XML souboru s časovými značkami

Závěr

Cílem mé bakalářské práce bylo analyzovat současnou nabídku řešení pro zpracování záznamu přednášek, porovnat metody pro porovnání dvou různých statických snímků a implementovat program, který bude v ukončeném záznamu obrazovky se slidy prezentace detekovat okamžiky, kdy došlo k přechodu na další slide.

Během samotného vyhledávání současných řešení jsem byl překvapen, kolik metod pro interaktivní záznam přednášek již existuje. Při implementaci metod pro porovnávání dvou rozdílných snímků jsem se přiučil mnoho nových věcí, které mi více přiblížili práci s videem na úrovni práce s kodekem, což bylo příjemnou změnou od mé běžné práce s videem v editačních programech.

Samotný výsledek praktické části splňuje zadání a tvoří fungující program pro analýzu záznamu přednášek. Různé algoritmy mají různé přednosti dle typu zaznamenané přednášky, avšak algoritmus $PSNR$ se ukázal jako nejvhodnější ke zpracování obecných přednášek nejen svoji spolehlivostí, ale i výpočetní nenáročností.

V závěru lze považovat cíle bakalářské práce za splněné.

Literatura

[1] Nagai, T.: Automated lecture recording system with AVCHD camcorder and microserver [online]. St. Louis, leden 2009, s. 47–54, doi: 10.1145/1629501.1629512. Dostupné z: https://www.researchgate.net/publication/221469065_Automated_lecture_recording_system_with_AVCHD_camcorder_and_microserver

[2] Wang, F.; Ngo, C.-W.; Pong, T.-C.: Synchronization of Lecture Videos and Electronic Slides by Video Text Analysis [online]. listopad 2013. Dostupné z: http://vireo.cs.cityu.edu.hk/papers/mm03-2.pdf

[3] Schroth, G.; Cheung, N.-M.; Steinbach, E.; aj.: Synchronization of presentation slides and lecture videos using bit rate sequences [online]. 2011. Dostupné z: https://web.stanford.edu/~bgirod/pdfs/Schroth_ICIP2011.pdf

[4] Sudhakaran, S.: Intra-frame vs Inter-frame Compression [online]. Dostupné z: https://wolfcrow.com/blog/intra-frame-vs-inter-frame-compression/

[5] SlidesLive - Professional Conference Recording [online]. [cit. 2018-06-14]. Dostupné z: https://slideslive.com/

[6] Mediasite Video Platform - Sonic Foundry [online]. [cit. 2018-06-27]. Dostupné z: http://www.sonicfoundry.com/mediasite/

[7] Colucci, E.: Image and Video quality assessment – Part One: MSE & PSNR [online]. duben 2011. Dostupné z: http://emanuelecolucci.com/2011/04/image-and-video-quality-assessment-part-one-mse-psnr/

[8] Colucci, E.: Image and Video quality assessment – Part Two: SSIM Index [online]. červenec 2011. Dostupné z: http://emanuelecolucci.com/2011/06/image-and-video-quality-assessment-part-two-ssim-index/

[9] pHash.org: Home of pHash, the open source perceptual hash library [online]. [cit. 2018-06-21]. Dostupné z: http://www.phash.org/

[10] Burkoň, R.: Záznam obrazovky z přednášky o Apakrychlích [online]. Jedná se o záznam přednášky 'Co je Apakrychle?', který proběhl v rámci Informatického večeru na FIT ČVUT. Přednášena je Bc. Tomáše Nováčkem a Ing. Michalem Valentou, Ph.D. Dostupné z: https://www.youtube.com/watch?v=PUzbanaw3rk

[11] Burkoň, R.: Testovací video s mezními případy - vysoký datový tok [online]. Dostupné z: https://www.youtube.com/watch?v=ayc7_c1mYd8

[12] Burkoň, R.: Testovací video s mezními případy - nízký datový tok [online]. Dostupné z: https://www.youtube.com/watch?v=ayc7_c1mYd8

[13] Burkoň, R.: Záznam obrazovky z přednášky o Apakrychlích [online]. [cit. 2018-06-14]. Dostupné z: https://drive.google.com/file/d/0B60_a0i3j8AFZE5aeVVRb085cEU/view

[14] Burkoň, R.: Testovací video s mezními případy - vysoký datový tok [online]. [cit. 2018-06-14]. Dostupné z: https://drive.google.com/file/d/1RKT-rCOS5Ma5qsbTCbDqCswJ9LQ8EBw2/view

[15] Burkoň, R.: Testovací video s mezními případy - nízký datový tok [online]. [cit. 2018-06-14]. Dostupné z: https://drive.google.com/file/d/14vE2zsSqQ1MtAgVTBd_M-uT18zKrTiTa/view

[16] About - OpenCV library [online]. [cit. 2018-05-10]. Dostupné z: https://opencv.org/about.html

[17] OpenCV 3.4.1 - OpenCV library [online]. leden 2018, [cit. 2018-05-02]. Dostupné z: https://opencv.org/opencv-3-4-1.html

[18] Trollope, R.: Open-Sourced H.264 Removes Barriers to WebRTC [online]. říjen 2013, [cit. 2018-05-03]. Dostupné z: https://blogs.cisco.com/collaboration/open-source-h-264-removes-barriers-webrtc

Dodatek A

Seznam použitých zkratek

BSD licence licence se svobodným šířením

MPEG LA MPEG Licensing Administration

OCR Optical Character Recognition (optické rozeznávání znaků)

OpenCV Open Source Computer Vision Library (open source knihovna pro práci s obrazem)

OS Operační systém

PSNR Peak-signal to noise ratio (špičkový poměr signálu k šumu)

QR code Quick Response kód

RGB barevný model používaný v počítačové grafice, zakládá se na 3 barevných kanálech (červená, zelená, modrá)

SSD Solid-state disk

SSIM Structural similarity

XML Extensible markup language (rozšiřitelný značkovací jazyk)

Dodatek B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `doc` ..... složka s dokumentací implementace
  - `index.html` ..... hlavní soubor s dokumentací implementace
- `exe` ..... adresář se spustitelnou formou implementace
  - `BP_detekce_slidu.exe` ..... zkompilovaná verze implementace pro OS Windows
  - `opencv_ffmpeg341_64.dll` knihovna pro čtení videa pomocí OpenCV
  - `opencv_world341.dll` .... knihovna OpenCV pro základní manipulaci s obrazem
  - `openh264-1.7.0-win64.dll` ... knihovna pro ukládání videa ve kodeku H264
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `BP_detekce_slidu.cpp` ..... zdrojový kód implementace
    - `BP_detekce_slidu.h` ..... hlavičkový soubor implementace
  - `test_files` ..... složka s odkazy na testovací soubory
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS