VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROHLÍŽENÍ ROZMĚRNÝCH RASTROVÝCH GEODAT ONLINE

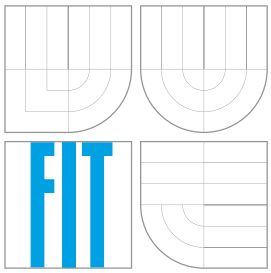
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN MIKITA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROHLÍŽENÍ ROZMĚRNÝCH RASTROVÝCH GEODAT ONLINE

VIEWING LARGE RASTER GEODATA ONLINE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN MIKITA

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce popisuje návrh multiplatformní klient-server aplikace pro přístup k rozměrným rastrovým datům, existující a navrhované způsoby optimalizace pro rychlost běhu a zpracování rastrů. Výhodou aplikace je aktualizovaná verze knihovny GDAL pro čtení většiny rastrových formátu a implementace rozšíření pro čtení GeoPDF pomocí open-source knihovny. Integrace vytvořené a optimalizované serverové části do open-source projektu IIPImage zpřístupní aktuálně nepodporované rastrové formáty tohoto projektu.

Abstract

This thesis describes the design of a multi-platform client-server application to access large raster data, existing and proposed ways to optimize the running speed and processing raster. The advantage of the application is an updated version of GDAL for reading most known raster format, and implementing extension for reading GeoPDF with open-source PDF rendering library. Integrating created and optimized server component into the open-source project IIPImage improves it to support more raster formats.

Klíčová slova

rozměrné rastrové data online, GDAL, Qt, PDFium, IIIF, IIPImage

Keywords

large raster geodata online, GDAL, Qt, PDFium, IIIF, IIPImage

Citace

Martin Mikita: Prohlížení rozměrných rastrových geodat online, diplomová práce, Brno, FIT VUT v Brně, 2015

Prohlížení rozměrných rastrových geodat online

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Mikita
27. května 2015

Poděkování

Na tomto místě bych rád poděkoval mému vedoucímu doc. Ing. Adamovi Heroutovi, Ph.D. za odborné vedení, za poskytnuté rady a konzultace. Chtěl bych také poděkovat konzultantovi Petru Přidalovi, Ph.D. za motivaci, za poskytnuté rady a informace potřebné k lepšímu pochopení této problematiky. V neposlední řadě bych rád poděkoval rodičům a kamarádům za projevenou podporu a trpělivost.

Obsah

Kapitola 1

Úvod

Většina grafických vstupů a výstupů vyžaduje kvalitní a detailní zpracování, které lze dosáhnout vysokým rozlišením obrazů. 2D grafické rastrové formáty podporují rozlišení až do $2^{31} - 1$ (více než 2 miliard pixelů pro šířku nebo výšku). Nicméně velmi vysoké rozlišení představuje pro běžného počítačového uživatele problém se zobrazením standardních formátu. Volně dostupné programy zobrazí rastr načtením nekomprimovaného obsahu do paměti a následným vykreslením. Pro rozlišení 25 000x25 000 formátu RGB to představuje 1,7 GiB dat potřebných k zpracování nebo uchování v paměti RAM. Zároveň zpracování i aplikace vyžaduje další místo.

Cílem diplomové práce je plně funkční multiplatformní aplikace, optimalizovaná na rychlost zpracování rozměrných rastrů a integrace systému do projektu IIPImage, který zobrazí všechny rastrové formáty podporované knihovnou GDAL. V první kapitole popisuji standardizované protokoly pro přístup k rozměrným datům (viz kapitola 2), které jsou motivací k náplni této práce. V kapitole 3 jsem otestoval projekty s otevřeným zdrojovým kódem, které se zaměřují na zobrazování rozměrných rastrů. Kapitola obsahuje jejich výhody a nevýhody, které jsem chtěl odstranit v rámci navrhované aplikace. Další částí zprávy je popis návrhu a implementace desktopové aplikace (viz kapitola 4). Součástí kapitoly je přidání podpory pro čtení GeoPDF souborů pomocí open-source knihovny PDFium. V poslední části kapitoly 4.5 jsem popsal integraci serverové části desktopové aplikace do projektu IIPImage. Kapitola 5 obsahuje způsob a výsledky testování rychlosti implementované aplikace a její optimalizaci. V poslední kapitole 6 jsem zapsal provedené experimentální testování a porovnání s existujícím řešením této problematiky.

Kapitola 2

Protokoly pro zpřístupnění rozměrných obrazových dat

Internetový prohlížeč zobrazuje obrazová data na požádání jednoduchým způsobem - načtením celého obrázku do paměti (nebo do cache) a zobrazením pomocí grafického rozhraní dané aplikace. Rozměrné obrazy mají nejen vysoké rozlišení, ale také velikost souboru je v řádech několika desítek až stovek MiB. Ani vysoká rychlost internetového připojení není dostatečná, aby zobrazila takový obrázek do pár vteřin. Navíc server může poskytovat menší rychlost přenosu, čímž omezuje uživatele.

V roce 1997 firmy Hewlett Packard, Live Picture a Eastman Kodak vytvořily první verzi **IIP protokolu - Internet Imaging Protocol** [10]. Konsorcium OGC (Open Geospatial Consortium) představilo v dubnu roku 2000 první verzi standardu **WMS - Web Map Service**. Od roku 2007 připravovalo nový standard **WMTS - Web Map Tile Service**, který byl oficiálně představen v roce 2010. Narůstající komunita světových vědeckých knihoven a obrazových úložišť zahájila spolupráci na vytvoření nové interoperabilní technologie pro poskytování obrazu [5]. V srpnu 2012 představilo první verzi protokolu **IIIF - International Image Interoperability Framework**.

Následující podkapitoly popisují výše představené protokoly a jejich specifikaci.

2.1 Internet Imaging Protocol - IIP

Podkapitola vychází z oficiální definice protokolu [2].

Poslední verze je 1.0.5, vydaná v říjnu 1997 (7 měsíců po první verzi) a nyní bez dalšího vývoje[1]. Protokol definuje efektivní přístup k rozměrným obrazům přes internetovou nebo intranetovou síť. Využívá výhodu formátu FlashPix, který byl představen stejnou skupinou organizací společně s firmou Microsoft Corporation v roce 1996. Formát FlashPix může obsahovat obrazy pro více rozlišení. Hierarchie rozlišení je seznam obrazů s poloviční velikostí pro šířku a výšku obrazu nad ním. Seznam končí rozlišením, které je stejné nebo menší než velikost dlaždice; pro formát FlashPix to je 64px pro šířku a výšku obrazu [1].

IIP protokol vraci jenom obraz jednoho rozlišení, které nejvíce odpovídá požadavku, čímž se snižuje datový přenos obrazu velkých rozměrů. Protokol je navržen jako nezávislá transportní vrstva. Část specifikace definuje speciální pokyny pro implementaci nad HTTP

[1] Kromě zdroje http://en.wikipedia.org/wiki/Internet_Imaging_Protocol není přístupná původní web stránka www.digitalimaging.org, která je citovaná z článku [10]

anebo TCP/IP schránkami. Komunikace IIP se skládá z požadavku na server a specifikace datových struktur posílané při této komunikaci (oboustranně).

2.1.1 Použití IIP v HTTP protokolu

Příkazy požadavku na server jsou složené z páru klíč-hodnota, které se zpracují zleva doprava, oddělené znakem ampersand (&). Odpověď je složená z jedné nebo více objektů, oddělené CRLF (HEX ASCII znaky kódu `0x0D0A`). Typ MIME v odpovědi může být `image/vnd.fpx` pro obrazy formátu FlashPix, `image/jpeg` pro obrazy formátu JPEG, nebo `application/vnd.netfpx` pro IIP objekty. Klient může používat metody `GET` nebo `POST`, přičemž požadavek je zakódován standardem `application/x-www-form-urlencoded`. Struktura URL je rozdílná pro integrovaný web-server (1.) a CGI server (2., název serveru FPXR). Pro integrovaný web-server je možné vynechat příkaz FIF s cestou k požadovanému obrazu, přičemž se získá ze samotné URL.

1. `http://adresa/dir1/dir2/file.fpx?OBJ=IIP,1.0&TIL=4,*`
2. `http://adresa/FPXR?FIF=/dir1/dir2/file.fpx&OBJ=IIP,1.0&TIL=4,*`

Příklad je ekvivalentní, nad souborem `/dir1/dir2/file.fpx` požaduje dlaždice pro rozlišení 4. úrovně, verze protokolu IIP 1.0.

2.1.2 Použití IIP pomocí TCP/IP schránek

Komunikace pomocí TCP/IP schránek není požadovaná pro server, ani pro klienta, ale musí podporovat sdělení této informace a záložní protokol pro zachování kompatibility. Příkazy by se nikdy neměly spojovat a posílat na jednom řádku, musí být oddělené CRLF (HEX ASCII znaky kódu `0x0D0A`). Stav trvá po dobu připojení. Pokud požadavek negeneruje žádnou odpověď, musí se poslat potvrzovací formule `OK\r\n` a ukončení odpovědi formulí `END\r\n`.

Úvodní připojení klienta na server by mělo začít pomocí HTTP komunikace na adresu s požadavkem `IIP-socket`. Server odpovídá objektem `IIP-socket` s IP adresou a portem, pokud podporuje tento typ komunikace. Server typu HTTP může odkázat na jiný server, podporující TCP/IP schránky.

2.1.3 Základní příkazy a objekty

Příkaz	Syntax	Popis
`FIF`	`FIF=`*`path`*	Název podporovaného obrazu
`OBJ`	`OBJ=`*`object`*	Žádost o objekt ze serveru
`TIL`	`TIL=`*`res,tile[,sub]`*	Žádost o jednu nebo více dlaždic s rozlišením
`SDS`	`SDS=`*`doid (,doid)*`*	Nastaví nový kořen datového úložiště
`JTL`	`JTL=`*`res,tile[,sub]`*	Vyžádá si celou dlaždici ve formátu JPEG

Tabulka 2.1: Seznam požadovaných základních příkazů.

Tabulka 2.1 definuje stručně základní a požadované příkazy pro komunikaci s IIP serverem. Tabulka 2.2 obsahuje vybrané objekty a jejich syntax pro komunikaci s IIP klientem. Při všech příkazech a objektech se nerozlišují velká a malá písmena.

Objekt	Syntax	Popis
`IIP`	`IIP:`*`version`*	Číslo verze protokolu - číslo s desetinnou čárkou
`IIP-server`	`IIP-server:`*`vendor(.capabs)*`*	ID poskytovatele a bitová maska podporovaných možností[a]
`Max-size`	`Max-size:`*`width height`*	Maximální velikost obrazu v pixelech
`Resolution-number`	`Resolution-number:`*`res`*	Maximální rozlišení (indexace od 1)
`Tile`	`Tile,`*`res,tile,sub/length:data`*	Jedna dlaždice obrazu

Tabulka 2.2: Výběr IIP objektů.

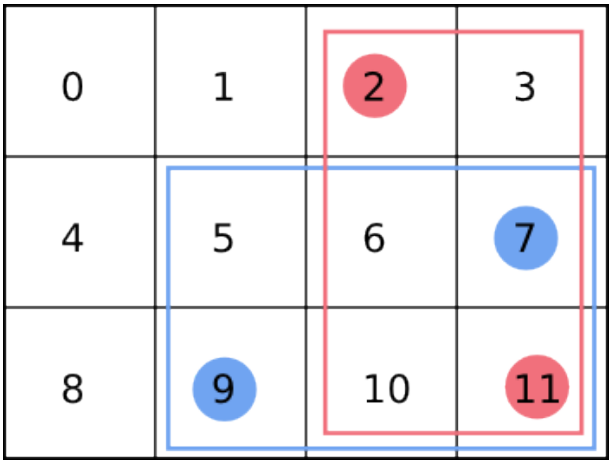
[a] Seznam a popis serverových možností je v části 3.3 specifikace[2]

2.1.4 Indexace dlaždic

Obrazový FlashPix formát souboru definuje dlaždicování a indexaci dlaždic takto:

- rozlišení obrazu R_ixC_i, rozlišení dlaždice R_txC_t,
- počátek obrazu je levý horní roh, souřadnice (0px, 0px),
- počet dlaždic v řádku je $N_R = \left\lceil \frac{R_i}{R_t} \right\rceil$,
- počet dlaždic ve sloupci je $N_C = \left\lceil \frac{C_i}{C_t} \right\rceil$,
- první index dlaždice je levý horní roh, souřadnice (0, 0),
- poslední index dlaždice je pravý dolní roh, souřadnice ($N_C - 1$, $N_R - 1$),
- jednočíselný index dlaždice začíná vlevo nahoře, pokračuje doprava, a pak z levé strany v dalším řádku, směrem dolů,
- jednočíselný index dlaždice (T_C, T_R) se vypočítá $T = T_R \times N_C + T_C$.

Při požadavku o rozsah dlaždic protokol definuje, aby rozsah byl seřazený matematickým způsobem, a výpočet dlaždic proběhne na serveru. Pro rozsah $2 - 7$ to jsou dlaždice: 2, 3, 6, 7, 10 a 11. Pro rozsah $7 - 9$ je výčet: 5, 6, 7, 9, 10 a 11.



Obrázek 2.1: Rozsah indexů dlaždic je matematicky seřazený

2.2 Web Map Service - WMS

Podkapitola vychází z oficiální definice protokolu [3].

Aktuální verze WMS protokolu je 1.3.0, vydaná v březnu 2006. Služba WMS vytváří mapy dynamicky z geografické informace[2]. Standard definuje "mapu", jako vyobrazení geografické informace v digitálním obrazovém souboru vhodném pro zobrazení na monitoru. Mezinárodní standard definuje tři operace:

1. **GetCapabilities** - vrací metadata o možnostech služby daného serveru,
2. **GetMap** - vrací mapu s dobře definovanými geografickými a rozměrovými parametry,
3. **GetFeatureInfo** - vrací informace o vlastnostech zobrazených na mapě.

Poslední operace je volitelná pro server a její implementace řadí server a klienta do druhé třídy shod - *Queryable WMS*. První třída, *Basic WMS*, vyžaduje implementaci prvních dvou operací a splnění požadavku *A.1* vypsaných v specifikaci protokolu v příloze A.

Hlavním požadavkem třídy *Basic WMS* je povinný parametr VERSION pro všechny požadavky kromě *GetCapabilities*. Verze je sloučená ze 3 kladných celých čísel, oddělené tečkou, $x.y.z$, kde y a z by měly být menší než 99. Pokud server podporuje více verzí, může s klientem vyjednávat, kterou verzi budou komunikovat.

1. Klient posílá (požadavek) největší podporovanou verzi protokolu.
2. Pokud server podporuje tuto verzi, odešle stejnou odpověď (stejné číslo verze).
3. Jinak server odešle nejvyšší podporovanou verzi, která je nižší než přijatá verze. Když server nepodporuje menší verzi, odešle nejmenší podporovanou verzi.
4. Pokud klient podporuje stejnou verzi, spojení je úspěšně navázáno a pokračuje v požadavcích s využitím této verze.
5. Když je verze větší než poslední poslaná verze vyjednávání neúspěšně skončilo a klient ukončí spojení.

[2] Geografická informace - týkají se jevů implicitně nebo explicitně spojených s umístěním vzhledem k Zemi. [3]

6. Jinak pošle menší podporovanou verzi, a pokračuje se krokem 2.

2.2.1 Implementace WMS pro HTTP protokol

Mezinárodní standard definuje implementaci pro platformu s podporou HTTP. Přístupy ke zdrojům jsou pomocí HTTP URL. Standard definuje pouze dotazovou (query) část URL pro službu WMS. GET metoda je povinná, na rozdíl od POST, ale struktura URL je definovaná zvláště pro obě metody.

Struktura URL pro metodu POST vyžaduje kompletní URL, která je validní podle IETF RFC 2396 [4]. Klient posílá požadavky v těle POST zprávy ve formátu XML. Při použití metody GET, klient sestaví řetězec název/hodnota páru spojené znakem &, která se připojí k URL prefixu definovaného pro server (nachází se v odpovědích na GetCapabilities požadavek). Níže je definovaný URL prefix pro metodu GET. Závorky [ ] představují 0 nebo 1 výskyt, a závorky { } představují 0 nebo více výskytu.

`http://host[:port]/path[?{name[=value]&}]`

2.2.2 Povinné a volitelné parametry operací

Seznam povinných a volitelných parametrů pro operace definující standard WMS ve formátu název=hodnota. Název parametrů je přesný dle specifikace. Přesná hodnota parametrů začíná velkým písmenem, jinak to je název, forma nebo typ hodnoty parametru (více v popisu). Povinné parametry jsou podtrženy.

Povinná operace **GetCapabilities** obsahuje parametry:

- `VERSION=verze` - volitelná verze protokolu, slouží pro vyjednávání,
- `SERVICE=WMS` - povinný parametr definující požadovanou službu serveru (server může podporovat více služeb),
- `REQUEST=GetCapabilities` - tato operace,
- `FORMAT=mime_type` - výstupní formát odpovědí, výchozí hodnota `text/xml` by měla být implementovaná pro každý server a klienta. Seznam jiných možných hodnot je v odpovědi na požadavek **GetCapabilities** v seznamu prvků *Request/GetCapabilities/Format*,
- `UPDATESEQUENCE=řetězec` - sekvence čísel nebo řetězců, slouží pro řízení vyrovnávací paměti.

Povinná operace **GetMap** obsahuje parametry:

- `VERSION=verze` - povinná verze protokolu (pro poslední revizi 1.3.0),
- `REQUEST=GetMap` - tato operace,
- `LAYERS=seznam` - čárkou oddělený seznam jedné nebo více vrstev mapy,
- `STYLES=seznam` - čárkou oddělený seznam stylu pro každou požadovanou vrstvu,
- `CRS=ns:id` - souřadnicový referenční systém (SRS) definovaný jmenným prostorem a identifikátorem,

- `BBOX=minx,miny,maxx,maxy` - rohy ohraničujícího rámečku v jednotkách SRS - levý dolní, pravý horní,
- `WIDTH=px` - výstupní šířka obrazu v pixelech,
- `HEIGHT=px` - výstupní výška obrazu v pixelech,
- `FORMAT=mime_type` - povinný výstupní formát, není definovaná výchozí hodnota. Pro HTTP protokol by to měla být hodnota typu MIME, která se použije v hlavičce odpovědí,
- `TRANSPARENT=TRUE|FALSE` - nastavení průhledné barvy pozadí mapy, výchozí hodnota FALSE (TRUE = ano, FALSE = ne),
- `BGCOLOR=barva` - barva pozadí ve formátu hexadecimálního čísla: 0xRRGGBB - červená, zelená, modrá; výchozí hodnota 0xFFFFFF,
- `EXCEPTIONS=formát` - formát oznámení výjimky, výchozí XML, jiné možné hodnoty jsou INIMAGE[3] a BLANK[4], které musí být v odpovědi **GetCapabilities** vyčteny,
- `TIME=čas` - časový údaj pro vrstvy,
- `ELEVATION=výška` - nadmořská výška vrstvy.

Volitelná operace **GetFeatureInfo** obsahuje parametry:

- `VERSION=verze` - povinná verze protokolu (pro poslední revizí 1.3.0),
- `REQUEST=GetFeatureInfo` - tato operace,
- požadavek z GetMap operace - kopie parametrů pro GetMap bez VERSION a REQUEST parametrů,
- `QUERY_LAYERS=seznam` - čárkou oddělený seznam jedné nebo více vrstev mapy[5],
- `INFO_FORMAT=formát` - povinný výstupní formát, typu MIME,
- `FEATURE_COUNT=počet` - volitelný maximální počet vlastností,
- `I=sloupec` - povinná souřadnice v pixelech v mapovém souřadnicovém systému,
- `J=řádek` - povinná souřadnice v pixelech v mapovém souřadnicovém systému,
- `EXCEPTIONS=formát` - volitelný formát oznámení výjimky, výchozí hodnota XML. Standard nedefinuje žádné jiné možné formáty výstupu.

[3] `EXCEPTION=INIMAGE` - výjimka bude vrácena v požadovaném výstupním formátu s přidáním textu popisující výjimku. V případě obrazového výstupu bude text vykreslený a vrácený obraz

[4] `EXCEPTION=BLANK` - výjimka bude vrácena v požadovaném výstupním formátu bez obsahu (prázdný obraz, žádná odpověď)

[5] Hodnota parametru `QUERY_LAYERS` se může lišit od hodnoty parametru `LAYERS` z požadavku **GetMap**

2.3 Web Map Tile Service - WMTS

Podkapitola vychází z oficiální definice protokolu [11].

Standard WMTS vychází z úsilí o vytvoření škálovatelné, vysoce výkonné služby pro distribuci kartografických map ve webovém rozhraní. Tato služba doplňuje dříve definovanou (a výše popsanou) službu WMS. Zatímco se WMS zaměřuje na vykreslování map pro dynamická data (získávání vlastností z mapy) anebo pro uživatelské stylizování map, služba WMTS je zaměřená na statická data s neměnnou strukturou dlaždic. Fixní sada dlaždic umožňuje lepší použití vyrovnávací paměti pro snížení síťové zátěže. Standard sdílí pojmy a koncepty se službou WMS ve verzi 1.3.0. Definice služby je zavedená pro snahu o přesun zpracování některých částí serveru na stranu klienta - libovolné uživatelem požadované transformace obrazu.

Mezinárodní standard definuje tři operace:

1. **GetCapabilities** - vrací metadata o možnostech služby daného serveru,
2. **GetTile** - vrací požadovanou dlaždici - část mapy,
3. **GetFeatureInfo** - vrací informace o vlastnostech zobrazených na mapě.

Poslední operace je volitelná pro server.

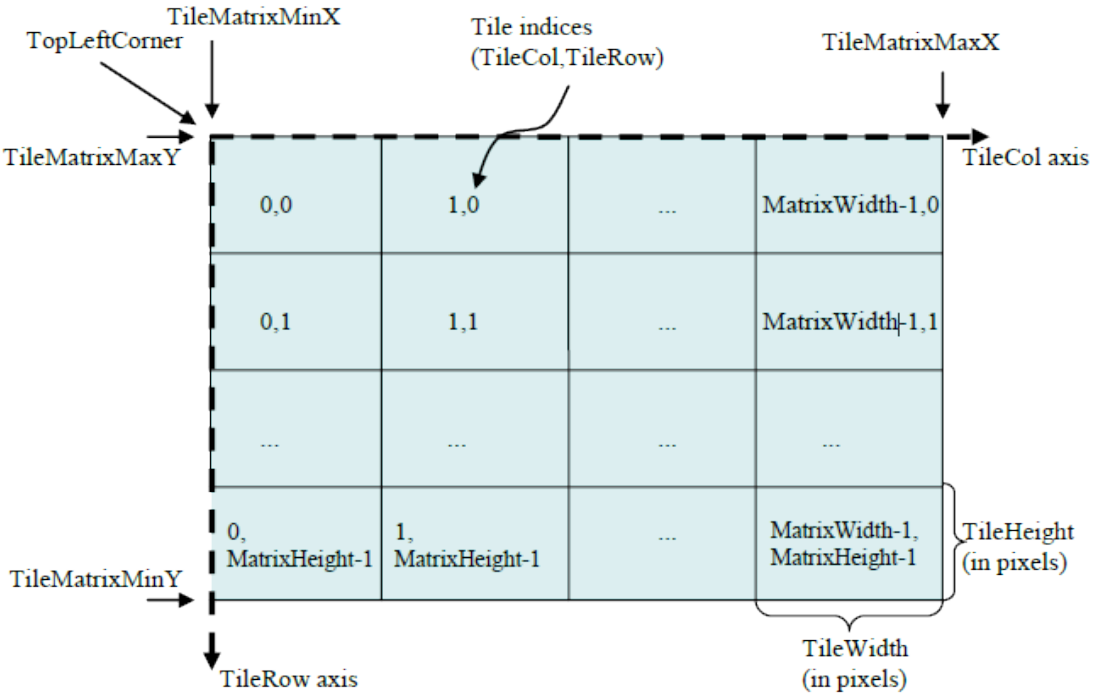
2.3.1 Sada matric dlaždic

Hlavním přínosem standardu WMTS je definice pojmů: dlaždice, matrice dlaždic a sada matric dlaždic. Souřadnicový systém obrazu má počátek (0px, 0px) v levém dolním rohu (stejně jako pro WMS), ale souřadnice dlaždic mají počátek v levém horním rohu (stejný způsob, který využívá FlashPix, viz sekce 2.1.4). Na obrázku 2.2 je levý horní roh označen (tileMatrixMinX, **tileMatrixMaxY**) a levý dolní roh (tileMatrixMinX, **tileMatrixMinY**).

- **Dlaždice** je definovaná souřadnicemi (TileCol, TileRow) s počátkem v levém horním rohu a index stoupá směrem doprava a dolů.
- **Matrice dlaždic** je definovaná pomocí:
  - unikátního identifikátoru typu URI (řetězec),
  - levého horního bodu (tileMatrixMinX, tileMatrixMaxY) v jednotkách SRS,
  - šířkou a výškou dlaždice (tileWidth, tileHeight) v pixelech,
  - šířkou a výškou matrice (matrixWidth, matrixHeight) v počtu dlaždic,
  - koeficientem převodu SRS jednotky na metry (metersPerUnit),
  - měřítkem (1:scaleDenominator).
- **Sada matric dlaždic** je definovaná pomocí:
  - unikátního identifikátoru typu URI (řetězec),
  - měřítkem (1:scaleDenominator),
  - souřadnicovým referenčním systémem (SRS),
  - ohraničujícím rámečkem – levý horní a pravý dolní roh v jednotkách SRS.

V některých jiných standardech je tento způsob dělení obrazu na dlaždice, za předpokladu, že jedna dlaždice je rozdělená rovnoměrně na 4 dlaždice následujícího měřítka, zvaný **obrazová pyramida**. Tento standard WMTS nedefinuje omezení na tento způsob dělení, proto jiné standardy mohou používat WMTS beze změny, ale obráceně to není zaručeno.

Vrstva mapy může definovat jednu nebo více takovýchto sad, s různým souřadnicovým systémem, ikdyž většinou každá vrstva mapy používá odkaz na stejnou sadu matric dlaždic.



Obrázek 2.2: Rozdělení prostoru na dlaždice, převzato ze standardu [11]

2.4 International Image Interoperability Framework - IIIF

Ani výše popsané či jiné existující protokoly nejsou dostatečné pro to, aby pokryly všechny novodobé požadavky na přístup k obrazovým datům, zejména pro rozměrná data. Proto byl v roce 2012 představen nový protokol, který má tyto hlavní cíle (převzato z [5]):

- zpřístupnit vědcům a studentům bezprecedentní úroveň a bohatý přístup k obrazovým zdrojům po celém světě,
- definovat soubor společných aplikačních programovacích rozhraní (API), které podpoří interoperabilitu mezi různými poskytovateli obrazů,
- vyvíjet, rozvíjet a dokumentovat sdílené technologie, jako obrazové servery a webové klienty, kteří poskytují světově-prvotřídní uživatelské zkušenosti v prohlížení, porovnávání, manipulaci a anotaci obrázků.

Poslední stabilní verze 2.0 byla představená v srpnu 2014. Oproti výše popsaným protokolům, standard IIIF definuje právě 2 typy požadavků na server:

- **Image request** - dotaz na obraz, nebo část obrazu,
- **Image information request** - dotaz na popis obrazu a možné funkcionality na tomto obrazu.

Protokol definuje přesnou a pevnou strukturu **URI**[6], která sdílí 4 parametry:

- **scheme** - schéma použitého protokolu, HTTP anebo HTTPS,
- **server** - server poskytující službu.
- **prefix** - volitelná cesta ke konkrétní službě, může obsahovat více segmentů oddělených lomítkem, ostatní znaky musí být zakódovány podle specifikace,
- **identifier** - identifikátor k přistupovanému obrazu.

Společná základní část URI požadavků:
`{scheme}://{server}{/prefix}/{identifier}`
Formát URI pro definované požadavky:

- **Image request** -
`{scheme}://{server}{/prefix}/{identifier}/{region}/{size}/{rotation}/{quality}.{format}`,
- **Image information request** -
`{scheme}://{server}{/prefix}/{identifier}/info.json`.

2.4.1 Parametry požadavku na obraz

Následuje výčet a popis parametrů, které jsou nutné pro správný dotaz na obraz. Z definovaného formátu URI to jsou: *region*, *size*, *rotation*, *quality*, *format*.

Region

Parametr určující požadovanou oblast. Pokud požadovaná oblast přesahuje zdrojový obraz, služba by měla[7] vrátit ořezaný obraz než vyplňovat prázdné místo. Pokud je řádek nebo výška nulové hodnoty anebo je oblast úplně mimo obraz, server by měl vrátit chybu číslo 400. Podporované formy hodnot jsou v tabulce 2.3.

`full`	kompletní obraz bez oříznutí
`x,y,w,h`	požadovaná zdrojová oblast v pixelech
`pct:x,y,w,h`	požadovaná zdrojová oblast v procentech

Tabulka 2.3: Možné formy hodnoty parametru Region - oblast

[6] URI - Uniform Resource Identifier, jednotný identifikátor zdroje.
[7] Protokol IIIF vymezuje a rozlišuje fráze **může** (may), **měl by** (should) a **musí** (must).

Size

Parametr určující rozměry vráceného obrazu, na které je požadovaná oblast zmenšená. Podporované formy hodnot jsou v tabulce 2.4. Pokud výsledná šířka nebo výška má hodnotu 0, server by měl vrátit chybu číslo 400. Server může podporovat škálování nad rozměry požadované oblasti.

`full`	zdrojová oblast není změněná
`w,`	zmenšení výsledné oblasti na požadovanou šířku, se zachováním poměru
`h,`	zmenšení výsledné oblasti na požadovanou výšku, se zachováním poměru
`pct:n`	zmenšení výsledné oblasti na n% šířky a výšky, se zachováním poměru
`w,h`	zmenšení výsledné oblasti na požadovanou šířku a výšku
`!w,h`	zmenšení výsledné oblasti na požadovanou šířku a výšku, tak aby se zachoval poměr a obraz byl uvnitř oblasti

Tabulka 2.4: Možné formy hodnoty parametru Size - rozměr

Rotation

Tento parametr definuje rotaci a zrcadlení (viz možné hodnoty). Pokud je hodnota rotace mimo povolený rozsah (0 - 360), server by měl vrátit chybu číslo 400. Služba by měla vyhovět všem požadavkům, i když výsledný obraz bude mít jiné rozměry než je požadováno parametrem **size**. Obraz by neměl být zmenšený jako výsledek operace Rotation. Pro hodnoty jiné než násobek 90 stupňů, je doporučeno pro klienta požadovat formát, který podporuje průhlednost. Tento standard nedefinuje rozhraní pro nastavení barvy pozadí. Hodnoty parametru jsou v tabulce 2.5.

`n`	počet stupňů po směru hodinových ručiček od 0 do 360.
`!n`	obraz je vertikálně zrcadlen a pak rotovaný jako výše

Tabulka 2.5: Možné hodnoty parametru Rotation - rotace

Quality

Parametr určuje typ vráceného obrazu, zda-li se jedná o barevný, ve stupních šedi nebo černo-bílý obraz. Hodnoty pro tento parametr jsou popsané v tabulce 2.6. Hodnota `default` existuje pro podporu nejnižšího stupně kompatibility implementace protokolu. Použitím této hodnoty může klient žádat o obraz, aniž by měl informace o typu obrazu.

`color`	obraz v plných barvách (rgb/rgba/..., podle formátu)
`gray`	obraz ve stupních šedi
`bitonal`	černo-bílý obraz (každý pixel je buď černý, nebo bílý)
`default`	serverová výchozí hodnota pro kvalitu obrazu (`color`, `gray`, `bitonal`)

Tabulka 2.6: Možné hodnoty parametru Quality - kvalita

Format

Parametr určující formát vráceného obrazu. Pokud žádaný formát není podporován, tak server by měl vrátit chybu číslo 400. V tabulce 2.7 je výčet hodnot podporovaných proto-

kolem IIIF a jejích hodnoty MIME.

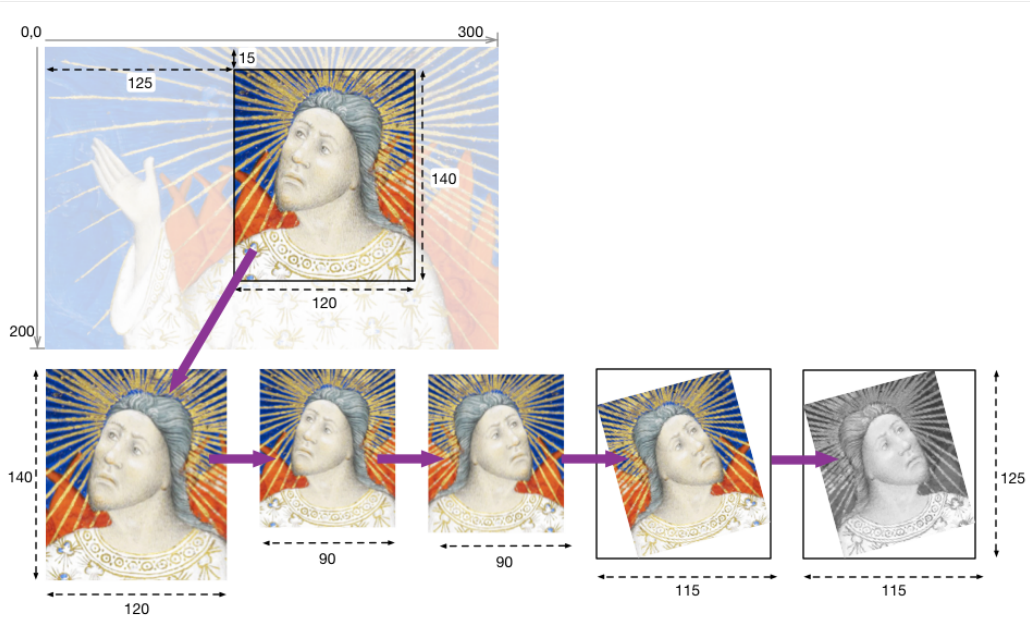
`jpg`	image/jpeg
`tif`	image/tiff
`png`	image/png
`gif`	image/gif
`jp2`	image/jp2
`pdf`	application/pdf
`webp`	image/webp

Tabulka 2.7: Možné hodnoty parametru Format - formát, typ MIME

2.4.2 Pořadí vyhodnocování parametrů

Protokol definuje přesné pořadí, v jakém se jednotlivé parametry vyhodnocují, což zabezpečuje pro stejné parametry požadavku stejný výsledek. Výše zmíněné protokoly neměly standardem stanovené pořadí, a proto výsledek záležel na implementaci a také na pořadí parametrů. V některých případech pořadí bylo nedefinované chování. Přesně definované pořadí operací, které odpovídají názvům parametrů:

1. Region,
2. Size,
3. Mirroring - pokud byl v parametru Rotation znak "!",
4. Rotation,
5. Quality,
6. Format.



Obrázek 2.3: Příklad správného vyhodnocování pořadí, převzato ze specifikace [17]

2.4.3 Vlastnosti odpovědi na požadavek informace o obrazu

Požadavek má právě jednu a přesně specifikovanou formu požadavku, viz úvod této sekce. Odpověď na tento požadavek obsahuje informace o obrazu samotném (rozlišení) a o podporovaných možnostech serveru (výstupní formáty). Specifikace vyžaduje, aby server odpověděl na žádostí tohoto typu (viz URI v úvodu sekce). Syntax odpovědi je definovaný podle JSON-LD[8]. Hlavička odpovědi typu `Content-Type` musí být `application/json` anebo `application/ld+json`.

V následujícím seznamu vlastností jsou požadované vlastnosti na nejvyšší úrovni kompatibility protokolu podtrženy:

- `@context` - musí to být URI `http://iiif.io/api/image/2/context.json`,
- `@id` - společná základní část URI pro obraz (se schématem, serverem, prefixem a identifikátorem),
- `protocol` - URI, který popisuje službu, v případě IIIF Image API to je `http://iiif.io/api/image`,
- `width` - šířka celého obrazu v pixelech,
- `height` - výška celého obrazu v pixelech,
- `profile` - pole profilů, které popisují podporované funkce serveru. První položka musí byt URI, které popisuje úroveň dodržování specifikace[9] ve formátu `http://iiif.io/api/image/<VERZE>/<ÚROVEŇ>.json`. Další prvky mohou být URI, anebo objekt s těmito vlastnostmi:

[8] JSON for Linking Data - http://www.w3.org/TR/json-ld/

[9] IIIF protokol pro Image API definuje 3 úrovně dodržování, viz http://iiif.io/api/image/2.0/compliance.html

  - `@context` - řetězec `http://iiif.io/api/image/2/context.json`,
  - `@id` - URI tohoto profilu,
  - `@type` - řetězec `iiif:ImageProfile`,
  - `formats` - pole podporovaných výstupních formátů (viz parametry požadavku),
  - `qualities` - pole podporovaných výstupních formátů (viz parametry požadavku),
  - `supports` - sada podporovaných funkcí, viz příloha A.1.

- `sizes` - pole párů `width` a `height`, které server podporuje při parametru `size` v dotazu na obraz,
- `tiles` - pole objektů, popisující dlaždicování pro pole měřítek, které server podporuje (dokáže efektivně zpracovat). Objekt má vlastnosti:
  - `scaleFactors` - pole měřítek, pro které platí velikost dlaždice,
  - `width` - šířka dlaždice,
  - `height` - výška dlaždice, pokud dlaždice není čtverec.
- `service` - vlastnost umožňuje přidat další vlastnosti obrazu, více informací na stránkách protokolu IIIF[10].

V příloze A.2 je příklad JSON odpovědi.

[10] Popis vlastností `service` protokolu IIIF http://iiif.io/api/annex/services/

Kapitola 3

Open-source projekty pro zobrazování rozměrný rastrových dat

Existuje několik různých aplikací a softvérových řešení, které mají snahu o rychlé zobrazení rozměrných rastrových dat. V této práci jsem se zabýval 3 projekty, které mají otevřený zdrojový kód a jsou známé mezi uživateli.

3.1 OpenEV2

Projekt **OpenEV** byl vyvíjen pro univerzity, vládní orgány, neziskové organizace a privátní firmy. Slouží jako aplikace pro zobrazování a analýzu geografických údajů, a zároveň jako knihovna pro vývojáře na vytváření nových aplikací. Poslední oficiální stabilní verze 1.8 je z roku 2004[1]. V roce 2012 vývojář Ko Nagase vytvořil veřejný zdrojový repozitář[2] s kopií projektu OpenEV a nazval ho **OpenEV2**. Pomocí uživatele Adam Fritzler[3] je aktuálně zdrojový kód kompilovatelný pro linuxovou distribuci Ubuntu 14.04.

Pro operační systém Windows je k dispozici souhrnný balíček **FWTools**[4], který mimo jiné obsahuje aplikaci OpenEV.

Vlastním testováním aplikace z právě zmíněného balíčku FWTools jsem dospěl k těmto výhodám a nevýhodám této aplikace.

- **Výhody:**
  - Nízká paměťová náročnost aplikace i pro rozměrné obrazy.
  - Rychlejší načítaní rozměrných obrazů vysokého rozlišení, než nativní prohlížeč obrázků operačního systému Windows *Windows Photo Viewer* od firmy Microsoft.
- **Nevýhody:**

[1] Seznam souborů na stránkách projektu: http://sourceforge.net/projects/openev/files/OpenEV/1.8.0/

[2] OpenEV verze 2 od Ko Nagase https://github.com/sanak/openev2/

[3] GitHub profile uživatele Adam Fritzler https://github.com/midendian

[4] Adresa balíčku FWTools http://fwtools.maptools.org/

- Manipulace s obrázkem je pro rozlišení víc než 100Mpx (10 000 x 10 000 px) zdlouhavá a časově náročná.
- Aplikace je jedno-vláknová - více vláken může zrychlit načítání obrázku.
- Grafické uživatelské rozhraní a přístup je pro dnešní dobu uživatelsky nepřívětivé (chybí přiblížování a pohyb obrazu pomocí myši).

3.2 VIPS/NIP2

Knihovna **VIPS** má dlouhou historii, která se datuje od roku 1989, kde vznikla jako systém pro zpracování obrazu k projektu VASARI[5], z čeho vznikl i název – *VASARI Image Processing System.*

Od roku 2005 spravuje knihovnu VIPS a její grafické rozhraní **nip2** John Cupitt působící na univerzitě Imperial College London, spolu s kolegy Kirk Martinez a Joe Padfield [7]. Knihovna VIPS je nyní standardní součástí většiny Linuxových distribucí. Grafické rozhraní **nip2** je něco mezi tabulkovou aplikací (Calc, Excel) a grafickým editorem (např. Photoshop). Různé transformace obrazu se zapisují jako vztahy mezi objekty (buňky tabulky), na základě kterých se vykresluje požadovaný výstup. Díky této technologii je jakákoliv transformace s rozměrným obrázkem rychlá, protože se uskuteční na požádání (zobrazení náhledu) a jenom pro část obrazu, která je viditelná na displeji[6].

Vlastním testováním aplikace pod operačním systémem Linux Ubuntu jsem dospěl k těmto výhodam a nevýhodam aplikace.

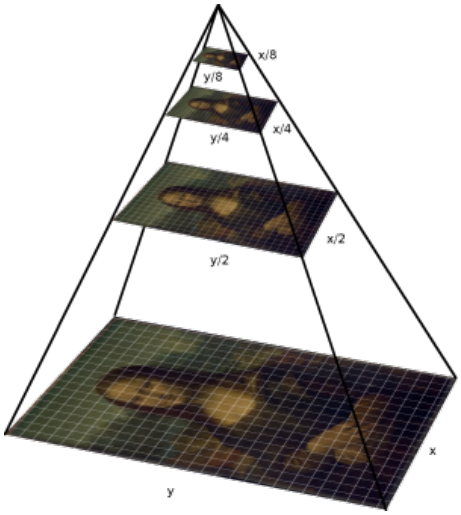
- **Výhody:**
  - Nízká paměťová náročnost aplikace i pro rozměrné obrazy.
  - Vícevláknové zpracování.
- **Nevýhody:**
  - Načítání obrazu s více než 200Mpx je časově zdlouhavé.
  - Aplikace nezpracuje geografické data typu MrSID, ECW, KAP, JPEG2000
  - Načítání PDF (GeoPDF) formátu je vítané, ale kvalita zpracování (renderování) je slabá až nevyhovující.
  - Pro rychlé prohlížení obrazů je zbytečně náročná a pomalá.

3.3 IIPImage

Podoba projektu **IIPImage** byla vytvořena již v roce 1996 na základě evropského projektu Viseum[7] a později použitý pro Acohir v roce 1999[8]. Cílem bylo vytvořit systém na prohlížení kolorimetrických obrazů velmi vysokých rozlišení, přístupných pomocí internetové sítě. Podklady k této kapitole jsou převzaty ze stránek projektu [12].

Projekt IIPImage je klient-server systém, který primárně poskytuje serverovou část a nabízí podporovanou klientskou část (webovou prohlížečku). Poslední stabilní verze IIPImage podporuje implementaci protokolu IIIF. Připravovaná verze 1.0 má implementaci

[5] Webová stránka projektu VASARI http://users.ecs.soton.ac.uk/km/projs/vasari
[6] Aplikace NIP2 http://www.vips.ecs.soton.ac.uk/index.php?title=Nip2
[7] Evropský projekt Viseum http://users.ecs.soton.ac.uk/km/projs/viseum/
[8] Projekt Acohir http://users.ecs.soton.ac.uk/km/projs/acohir/



Obrázek 3.1: Podporovaný typ souboru TIFF vyžaduje pyramidu dlaždic. Převzato ze stránek projektu.

IIIF verze 2.0 s plnou první úrovní dodržování (level 1 compliance). Serverová část vrací na požádání dlaždice ve formátu JPEG.

Vlastním testováním jsem si potvrdil informaci o projektu, který podporuje jenom vstupní soubory typu **JPEG2000***[9] a **TIFF**, který musí být složený z dlaždic a sestavený do pyramidy s vícero rozlišení[10]. Samotný projekt v dokumentaci poskytuje návod na konverzi souboru do požadovaného formátu pomoci knihovny VIPS (viz sekcí 3.2).

Serverová část je optimalizovaná na rychlost zpracování, např. využitím více vláken (každé vlákno zpracuje jednu žádost). Jedinou nevýhodou je slabá podpora formátů vstupních souborů. Jedním z cílů této práce je odstranění tohoto nedostatku integrací vytvořené a optimalizované části aplikace, která je popsaná v následující kapitole.

[9] Podpora formátu JPEG2000 v IIPImage projektu požaduje knihovnu Kakadu pro dekódování, která není open-source a projekt IIPImage nemá licenci pro distribuci této knihovny.

[10] Podporované formáty obrazů serverem IIP Image http://iipimage.sourceforge.net/documentation/images

Kapitola 4

Aplikace QtGDAL

Jedním z hlavních bodů práce je návrh a implementace multiplatformní aplikace, která pomocí klient-server technologie umožní rychlé prohlížení rozměrných rastrových dat. Aplikace používá knihovnu GDAL[1] pro čtení obrazových souboru, která vyniká množstvím podporovaných rastrových formátů[2] a jejich rychlým zpracováním s nízkým nárokem na paměť.

Zdrojový kód aplikace je rozdělen na serverou část ve složce `src/` a grafické rozhraní aplikace ve složce `gui/`. Serverová část je nezávislá na grafické, pro lepší nezávislou údržbu, případně pro využití v jiných projektech (integrace do IIPImage).

V této kapitole popisuji implementační detaily serverové části aplikace a její grafické rozhraní. Pro experimentální testování a vyhodnocení výsledné aplikace jsem implementoval speciální testovací rozšíření, které je popsané v podkapitole 4.3. V následujících podkapitolách jsem popsal postup kompilace knihovny PDFium a její implementaci v rámci GDAL GeoPDF zásuvného modulu. Poslední podkapitola obsahuje implementační detaily pro integraci serverové části aplikace QtGDAL do serveru IIPImage.

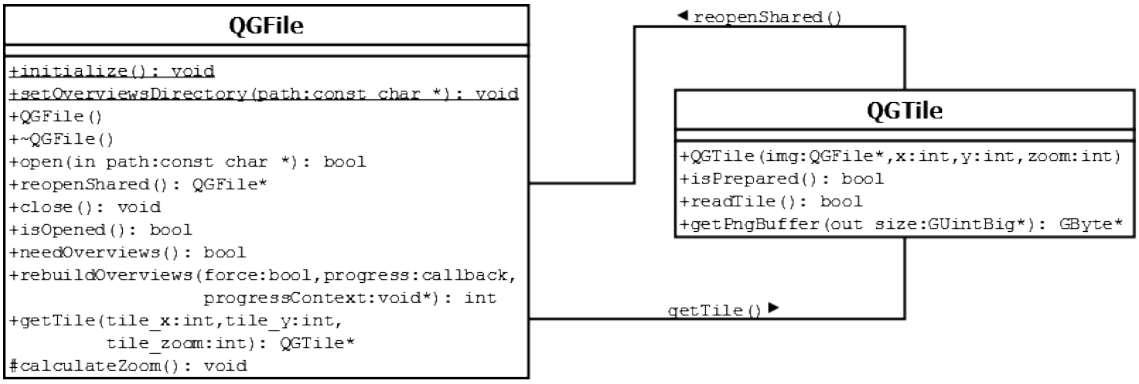
4.1 Serverová část zdrojového kódu

Serverová část je napsaná v jazyce **C++** ve standardu C++11, s využitím OOP (Object Oriented Programming) paradigmatu a modelu tříd. Hlavní částí serveru jsou třídy:

- `QGFile` - soubor `file.cpp` a
- `QGTile` - soubor `tile.cpp`.

Server je připraven pracovat s více soubory současně, přičemž každá instance třídy `QGFile` vytváří a využívá objekty třídy `QGTile` (viz obrázek 4.1).

[1] Geospatial Data Abstraction Library http://gdal.org/1.11/index.html
[2] Rastrové formáty GDAL http://gdal.org/1.11/formats_list.html



Obrázek 4.1: Diagramy třídy `QGFile` a `QGTile` a jejich důležité metody a navázaní mezi nimi.

Třída `QGFile` otevírá vstupní soubor a připravuje objekt na požadavky na dlaždice, zároveň poskytuje metodu `getTile(x, y, zoom)`, která vrací instance třídy `QGTile`. Parametry funkce jsou souřadnice dlaždice a úroveň přiblížení. Souřadnice dlaždice odpovídají standardu WMTS (viz kapitola 2.3) Velikost dlaždice je pevná 256 x 256 px.

Konstruktor třídy `QGTile` nejdříve požádá o vytvoření sdílené kopie objektu `QGFile`, která umožňuje vícevláknový přístup k otevřenému souboru. Pak vypočítá koordinace vyžádané dlaždice obrazu (pozici a čtenou velikost dlaždice) a připraví dočasné paměťové soubory pro data pixelů a výsledný PNG soubor. Metoda `readTile()` implementuje samotné načtení části obrazu do vyrovnávací paměti. Po úspěšném načtení se volá metoda `getPngBuffer(size*)`, která převede pixelové data (ve formátu RGBA, paleta nebo jiné) na formát PNG.

Načítání části obrazu pro požadovanou dlaždici se uskutečňuje v následujících fázích:

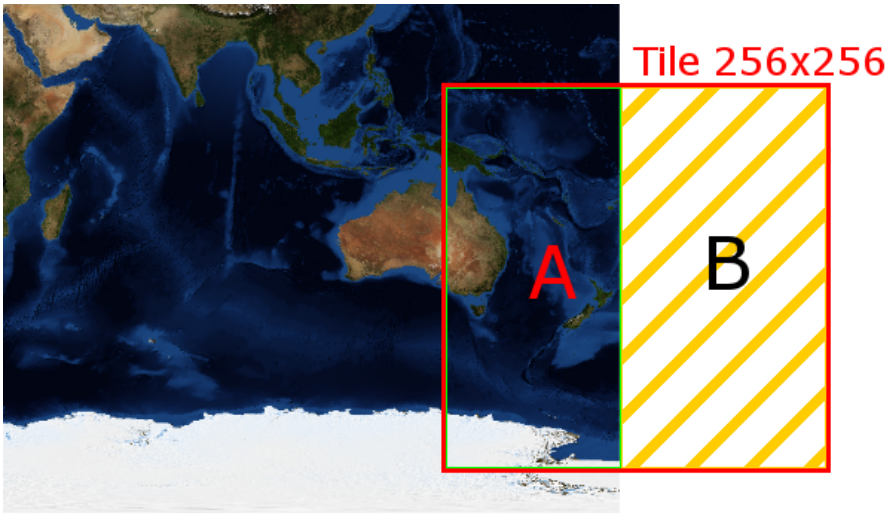
1. detekce částečné dlaždice,
2. příprava počtu kanálu dlaždice,
3. alokace a inicializace bufferu,
4. čtení pixelů z obrazu.

1. Detekce částečné dlaždice nastaví správné hodnoty pro čtení zdrojového obrazu, aby se nečetly pixely, které se nenachází v obrazu (a předešlo se pádům aplikace). Detekce zachovává poměr mezi požadovanou velikostí a možnou velikostí obrazu.

2. Počet kanálů výsledné dlaždice je upraven přidáním alfa kanálu (průhlednost). Je to možné ale jen pro určité typy formátu obrazu. Například obraz s paletou barev nemůže mít průhledný kanál, protože zpracování takového typu počítá s průhlednou barvou v paletě barev. Prohledávání a zjišťování barev by bylo časově zbytečné prodloužení, které by nevedlo k adekvátní kvalitě. Pokud je obraz ve stupních šedi, tak průhledný kanál je druhý v pořadí, jinak bude čtvrtý v pořadí.

3. Výstupní buffer je inicializován tak, aby nastavil bílé pozadí a neprůhlednost pro čtenou oblast (část A na obrázku 4.2), a v případě přidaného kanálu průhlednosti je nastavena hodnota na plnou průhlednost jen v oblastech mimo čtenou oblast (část B na obrázku 4.2).

4. Čtení pixelů z obrazu podle požadované dlaždice se provede voláním funkce knihovny GDAL `GDALDataset::RasterIO()`, které jsou předány připravené argumenty a inicializovaný buffer pro čtení oblasti (část A na obrázku 4.2).



Obrázek 4.2: Oblasti požadované krajní dlaždice obrazu.

Výsledná dlaždice ve formátu PNG je vytvořená pomocí dočasného paměťového souboru, který je naplněn barvami pixelů v nezpracovaném formátu, a který je načten z funkce `RasterIO()`. Pokud je potřeba, je zkopírovaná tabulka barev (pro paletový formát vstupního obrazu) a jsou nastavené interpretace každého použitého kanálu podle vstupního souboru. V případě přidaného kanálu průhlednosti je jeho interpretace patřičně nastavená. Z tohoto paměťového souboru je vytvořena kopie pomocí GDAL ovladače formátu PNG (jediný způsob vytvoření PNG souboru). Dočasný paměťový soubor je odstraněn a metoda vrací ukazatel na data souboru PNG v paměti.

4.2 Grafické rozhraní aplikace

Grafické rozhraní aplikace je postaveno na knihovně Qt verzi 4.8[3]. Aplikace obsahuje jediné hlavní okno, které zobrazuje obraz pomocí webového prohlížeče OpenLayers 3[4] s podporou protokolu IIIF, který je využitý pro zjednodušení URL požadavků.

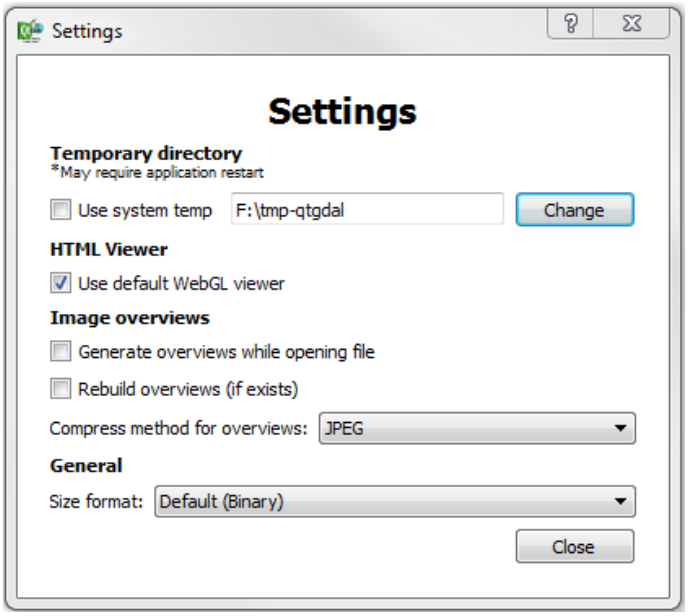
Webový prohlížeč ve formě samostatného HTML souboru (nevyžaduje přístup k internetové síti) je zobrazen pomocí knihovny QtWebKit (pod Windows OS a Linux) anebo pomocí nativního WebKit v operačním systému Mac OS X.

Pomocí menu anebo standardní klávesové zkratky pro otevírání souboru si uživatel vybere podporovaný rastrový obraz. Aplikace připraví server (otevření souboru v objektu `QGFile`) a načte HTML prohlížečku, kterou inicializuje na vlastností souboru - šířka a výška obrazu a maximální úroveň přiblížení. Webový prohlížeč posílá požadavky na dlaždice pomocí adresy tvaru:

```
gdal://file/zoom/x/y/png
```

Implementace `gdal` protokolu je pro QtWebKit a Mac WebKit oddělená, ale společná pro systém zpracování. K samotnému zpracování a vrácení dlaždice slouží třída **TileReader**, kterou je možné spouštět v samostatných vláknech.

[3] http://qt-project.org/doc/qt-4.8/
[4] http://openlayers.org/



Obrázek 4.3: Dialogové okno nastavení aplikace QtGDAL.

Grafické rozhraní poskytuje dialogové okno nastavení (viz obrázek 4.3) s možnostmi:

- **Cesta k dočasné složce** - slouží pro generování náhledu, pokud je otevřený soubor ve složce určené pouze na čtení.
- **WebGL** - možnost zapnout/vypnout testování a případné spuštění WebGL podpory.
- **Generování náhledů** - nastavení kompresní metody a možností generovat nebo vynutit generování náhledů.
- **Formát velikosti** - formát zobrazování velikostí souborů. Umožňuje přepnutí mezi *binárním* (1024 B = 1 KiB) formátem a *dekadickým* (1000 B = 1 KB) formátem.

4.3 Testovací rozšíření

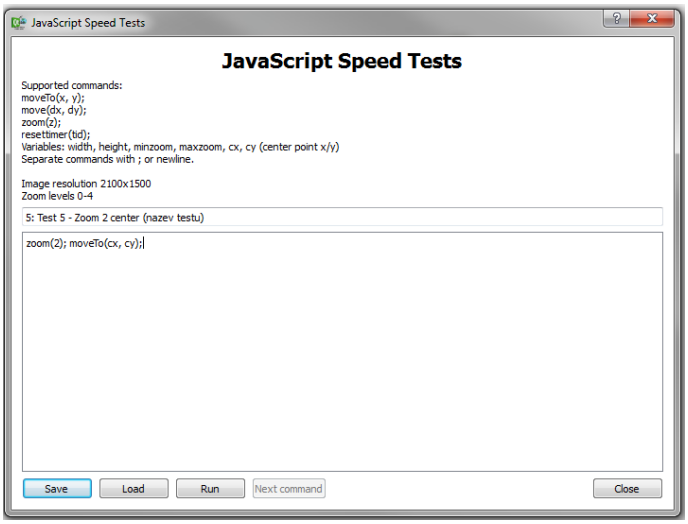
Pro účely testování aplikace jsem vytvořil speciální testovací rozšíření, které obsahuje dialogové okno JavaScriptových testů (viz obrázek 4.4) a okno časoměr (viz obrázek 4.5). Tohle rozšíření je kompilováno pouze při povolení konfigurace `tester` v projektovém souboru.

JavaScriptové testy podporují speciální příkazy, které jsou zpracované a upravené aplikací pro aktuální soubor. Výčet příkazů s krátkým popisem je v tabulce 4.1. Testy podporují pro tyto příkazy také proměnné, které jsou nahrazené hodnoty podle aktuálního souboru. Následující proměnné jsou samovysvětlující:

- `width` - šířka,
- `height` - výška,
- `minzoom`,
- `maxzoom`,
- `cx` - centrální bod x (polovina šířky),

- `cy` - centrální bod y (polovina výšky).

Testovací menu obsahuje položku s povolením libovolného JavaScript kódu (*Allow custom JavaScript*), který bude vykonán bez ověřování. Dialogové okno umožňuje uložení a načítání testů do/ze souboru. Tlačítko *Run* spustí aktuální test ve velkém textovém poli.

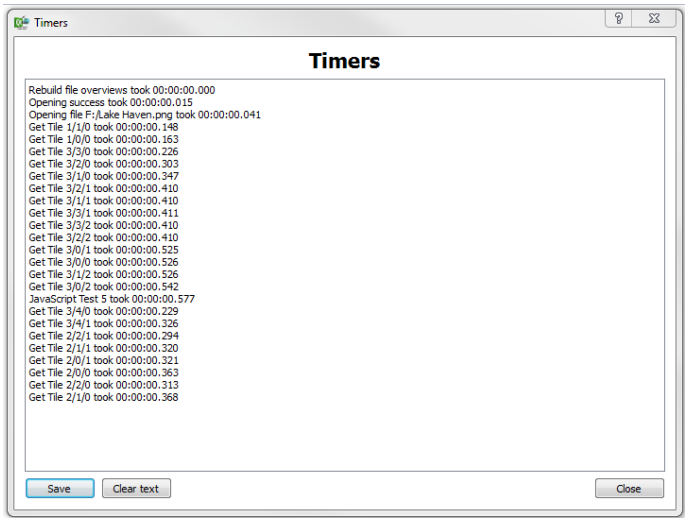


Obrázek 4.4: Dialogové okno JavaScriptových testů aplikace QtGDAL.

Celý test je rozdělen na řádky, přičemž v jednom řádku může být více příkazů oddělených středníkem. Jeden řádek testů se považuje za pod-test, který je zpracován okamžitě. Výsledek pod-testu by měl mít za následek změnu prohlížečky (pozice nebo úroveň přiblížení), která spustí načítání nových dlaždic obrazu. Po načtení všech dlaždic, které byly vyvolané touto změnou, je pod-test ukončen a vyvolá zpracování následujícího řádku testu. Pokud byl test ukončen celý (není žádný další řádek), pak je časový úsek mezi spuštěním testu a poslední načtenou dlaždicí zapsán jako výsledný čas testu do dialogového okna časoměr. Speciální případ je pro příkaz `resettimer(tid)`, který předčasně ukončí test s vypsáním času a spustí další časoměr pro následující příkazy a přidá k aktuálnímu číslu testu identifikátor z argumentu příkazu.

Příkaz	**Popis**
`moveTo(x, y)`	Přesunutí středu prohlížečky na pozici x, y
`move(dx, dy)`	Posun pozice středu prohlížečky o dx, dy
`zoom(z)`	Nastavení úrovně přiblížení
`resettimer`	Ukončení testů a spuštění nové časomíry pro zbytek testu

Tabulka 4.1: Seznam speciálních testovacích JavaScript příkazů.



Obrázek 4.5: Dialogové okno časoměr aplikace QtGDAL

Do dialogového okna časoměr se zapisují časy zpracování každé vyžádané dlaždice, případně spuštěných testů. Při každém otevření souboru se taktéž zapíše čas tvorby náhledů a celkový čas otevření souboru. Text v tomto okně je možné uložit do souboru a nebo ho smazat. Mazání textu není implicitní (tudíž po uložení se automaticky nesmaže).

Součástí testovacího rozšíření je zpracování argumentů při spuštění aplikace z terminálu. Argumenty umožňují spuštění automatických testů nad souborem, uložení výsledních časů a případné ukončení aplikace (v případě jednorázového testu).

Argument	**Popis**
`-autotestfile <path>`	Načtení automatického testu ze souboru
`-autotestin <string>`	Nastavení automatického testu na řetězec
`-autotestcount <number>`	Počet opakování testu
`-autotestout <path>`	Uložení výsledních časů testu
`-autotestpause <ms>`	Pauza v milisekundách mezi testy a před spuštěním
`-autotestquit`	Ukončení aplikace po skončení celého testu nad souborem

Tabulka 4.2: Seznam testovacích argumentů pro příkazový řádek.

Pro experimentální testování (viz kapitola 6) jsem rovněž připravil skripty pro Unix OS (Linux a Mac OS X) a Windows OS. Skript prochází testovací sadu a pro každý formát spustí aplikaci QtGDAL s testem ze souboru a počtem 5 iterací testu. Nejdříve se testuje natívní obrázek a poté se testuje rychlost s vytvářením náhledů. Více o testování je popsáno v kapitole 5.

4.4 Rozšíření GDAL o čtení GeoPDF pomocí PDFium

Knihovna GDAL implementovala podporu pro čtení PDF formátu ve verzi 1.8.0 s využitím knihovny poppler [6]. Od verze 1.9.0 je přidaná podpora knihovny PoDoFo [14] pro parsování informací o georeferencování. Důvodem byla licence GPL knihovny poppler - nutnost zachování open-source aplikace, která jí využívá. Nicméně pro rasterizaci PDF je nutné mít dostupnou aplikaci `pdftoppm`, která je součástí distribuce poppler. Licence PoDoFo je

LGPL[5], která dovoluje použití sdílené knihovny v proprietarních aplikacích s uzavřeným kódem. Volání procesu `pdftoppm` je v souladu s licencí GPL[6], ale není to čisté řešení. To bylo hlavním důvodem, proč upravit podporu GeoPDF v knihovně GDAL použitím open-source knihovny **PDFium**, která je pod licencí BSD New[7].

PDFium [16] je renderovací engine formátu PDF používaný v prohlížeči Chromium. Základní zdrojové soubory poskytla firma Foxit Software Inc. ve spolupráci s firmou Google, která je vydala pod open-source licencí, a využila pro svůj prohlížeč Chrome[8]. Od června 2014 [8] je pak PDFium vyvíjen a upravován programátory prohlížeče Chromium [15].

Následující sekce popisují přesné kroky k docílení podpory PDFium v knihovně GDAL ve verzi 1.11.2[9]. Nejdříve jsou uvedené a popsané změny ve zdrojových kódech knihovny PDFium a GDAL, následně postup kompilování pro platformy Windows, Linux a Mac OS X.

4.4.1 Knihovna PDFium

Knihovna PDFium je úzce svázaná s knihovnou V8 JavaScript Engine [9]. V8 Engine je datově objemný a časově zvyšuje dobu kompilace PDFium knihovny, přičemž není potřebná pro rasterizaci GeoPDF, proto je možné ji odstranit pomocí následujících postupů.

Základem následujících úprav je verze PDFium ze dne 7. 2. 2015, hash commitu 44fc192[10]. Původním autorem některých změn pro možnost vypnutí podpory V8 JavaScript Engine je Adam Schepis, který je provedl na starším kódu PDFium a současně přidal podporu pro kompilaci na iOS [13].

Soubor `fpdfsdk/src/fsdk_mgr.cpp` vyžaduje přidání makro-hlídky na `_V8_SUPPORT_` pro funkce:

- `GetJSRuntimeFactory()` - řádek 214,
- `CPDFDoc_Environment::CPDFDoc_Environment(CPDF_Document* pDoc)` - řádek 233-237,
- `CPDFDoc_Environment::~CPDFDoc_Environment()` - řádek 245-251,
- `CPDFDoc_Environment::GetJSRuntime()` - řádek 272-280,

Soubor `pdfium.gyp` vyžaduje přidání proměnné `pdf_use_v8%: 0,` na 3. řádek. Přednastavená hodnota značí o vypnutí podpory V8. Na konci části `variables` (řádek 13), je potřeba přidat podmíněné vložení konfiguračního souboru `javascript.gypi`, a do části `target_defaults` vložit podmíněné nastavení makra `_V8_SUPPORT_`. Na řádku 72 je potřeba odstranit závislosti (dependencies) `javascript` a `jsapi` a přesunout do samostatné podmíněné části na základě proměnné `pdf_use_v8`. Poslední změnou je přesunutí dvou cílů (target) `javascript` a `jsapi` do samostatného souboru `javascript.gypi`.

Pro korektní vytvoření kompilačních souborů je nutné ještě upravit soubor `samples/samples.gyp` přidaním proměnné `pdf_use_v8%: 0` a přesunutím závislosti a vkládaných cest z V8 JavaScript Engine složky `v8/` do podmíněné části (viz zdrojový kód B.4).

[5] LGPL licence http://www.gnu.org/copyleft/lesser.html
[6] GPL licence, verze 2 https://www.gnu.org/licenses/gpl-2.0.html
[7] BSD 3-Clause licence http://opensource.org/licenses/BSD-3-Clause
[8] Google Chrome je prohlížeč s uzavřeným kódem, který využívá open-source Chromium
[9] GDAL 1.11.2 vydaná v únoru 2015 http://trac.osgeo.org/gdal/wiki/DownloadSource#a1.11.2LatestStableRelease-February2015
[10] Zdrojový kód PDFium ke stažení https://pdfium.googlesource.com/pdfium/+/44fc192f29a77c5864fabffe5ab63937dacdfd21/

```
 CJS_RuntimeFactory* GetJSRuntimeFactory()
 {
+#ifdef _V8_SUPPORT_
  static CJS_RuntimeFactory s_JSRuntimeFactory;
  return &s_JSRuntimeFactory;
+#else
+    return NULL;
+#endif
 }
```

Zdrojový kód 4.1: Ukázka změn v souboru `fpdfsdk/src/fsdk_mgr.cpp`

```
   'variables': {
     'pdf_use_skia%': 0,
+    'pdf_use_v8%': 0,
  ...
+  'conditions': [
+    ['pdf_use_v8==1', {
+      'includes': [
+        'javascript.gypi'
+      ]
+    }]
```

Zdrojový kód 4.2: Ukázka změn v souboru `pdfium.gyp`

Všechny zde popsané změny, jsou uvedené v příloze B.1 jako rozdílový soubor.

4.4.2 Multiplatformní kompilace PDFium

Kompilování knihovny PDFium je postaveno na projektu GYP[11], který na základě popisného souboru (viz soubor `pdfium.gyp` výše upravovaný) vytvoří soubory specifické pro kompilaci na dané platformě.

Nejdříve je potřeba stáhnout balíček `depot_tools`[12] a přidat jeho cestu do systémové proměnné `PATH`. Pak stáhneme zdrojové kódy systému GYP[13] a nainstalujeme pomocí python skriptu `python setup.py install`. Instalování vyžaduje administrátorská práva (na systémech UNIX, práva uživatele root).

Vytvoření platformově-specifických souborů pro kompilování je stejné na všech platformách použitím příkazu `python build/gyp_pdfium.py`.

Windows OS

Při testování různých nastavení pro korektní kompilování jsem narazil na problém se sdílenou knihovnou (DLL), která se využívá ve vícevláknovém kódu. Některé globální proměnné nebyly přístupné z jiných vláken. Samotná knihovna PDFium není připravená pro kompilaci sdílené knihovny, proto je doporučeno ponechat kompilaci statické knihovny. Popisný soubor projektu PDFium má nastavenou společnou složku výstupu pro 32-bitový a 64-bitový výstup, proto je doporučeno změnit položku `OutputDirectory` v nastavení `msvs_configuration_attributes` v souboru `build/standalone.gypi`, přidaním řetězce

[11] Generate Your Project https://code.google.com/p/gyp/
[12] Balíček `depot_tools` http://www.chromium.org/developers/how-tos/install-depot-tools
[13] Zdrojové kódy systému GYP https://chromium.googlesource.com/external/gyp

```
 {
+  'variables': {
+    'pdf_use_v8%': 0,
+  },
   'target_defaults': {
```

Zdrojový kód 4.3: Ukázka změn v souboru `samples/samples.gyp`

'`-$(PlatformName)`' na konec aktuální hodnoty. Další žádaná změna pro správnou kompilaci pod Windows OS je vypnutí hlášení varování jako chyb, které přeruší proces kompilování. Je to položka `WarnAsError` také ve výše zmíněném souboru.

V základním zdrojovém kódu PDFium, ze kterého jsem vycházel, je chyba při kompilaci, kdy se nastavila hodnota makra `OPJ_HAVE_INTTYPES_H`, která vkládá soubor, který se pod Windows OS s využitím prostředí Microsoft Visual Studio 2010 C++ Express nenachází. Chyba byla nahlášena s opravou[14].

Vývojové prostředí Visual Studio 2010 je více agresivní a vyžaduje potlačení varování pro kódy 4715 a 4244, navíc k již existujícím v souboru `pdfium.gyp` do proměnné `msvs_disabled_warnings`.

Všechny změny zde popsané, jsou uvedené v příloze B.2 jako rozdílový soubor.

Spuštění samotné kompilace je možné v grafickém prostředí Visual Studio 2010 po otevření souboru řešení `build/all.sln`, anebo použitím následujícího příkazu v příkazovém řádku vývojového prostředí `Visual Studio x64 Win64`:

```
$ msbuild build\all.sln /t:pdfium /p:Configuration=Release /p:Platform=x64
```

Pro kompilaci 32-bitového výstupu je potřeba změnit hodnotu `Platform` na `Win32`, a spuštění v konzole `Visual Studio Command Prompt` (bez přídavného x64). Výstupní soubory statické knihovny (LIB) se nachází ve složce `build/Release/lib`, případně ve složce `build/Release-x64/lib`, pokud se provedla změna výstupní složky zmíněné výše (v případě 32-bitového výstupu to bude `Release-Win32`). Počet souborů by měl být 13, a všechny je potřeba odkázat při kompilaci aplikace, která bude používat knihovnu PDFium (viz níže v sekci o GDAL GeoPDF pluginu).

Linux

Kompilace na OS Linux je možné zjednodušit změnou typu kompilace cíle `pdfium` na `shared_library` (řádek 59 souboru `pdfium.gyp`).

```
      'target_name': 'pdfium',
-     'type': 'static_library',
+     'type': 'shared_library',
      'dependencies': [
```

Zdrojový kód 4.4: Ukázka změny v souboru `pdfium.gyp` pro kompilaci na OS Linux

Po spuštění příkazu `$ ./build/gyp_pdfium` se vytvoří `Makefile` soubor, který se jednoduše spustí příkazem:

```
$ make BUILDTYPE=Release pdfium
```

[14] Nahlášená chyba v PDFium https://code.google.com/p/pdfium/issues/detail?id=73

Výstupní soubor je dynamická knihovna `libpdfium.so`, který se nachází ve složce `out/Release/lib.target`.

Mac OS X

Pro operační systém Mac OS X je vytvořen projektový soubor vývojového prostředí Xcode. Vlastním testováním doporučuji aplikovat změnu pro kompilaci na OS Linux (viz zdrojový kód 4.4). Otevřením tohoto souboru je možné zkompilovat knihovnu PDFium pomocí:

```
$ xcodebuild -configuration Release_x64 -target pdfium
```

Výstupní soubor se nachází ve složce `xcodebuild/Release_x64/lib.target`, je to dynamická knihovna `libpdfium.dylib`.

4.4.3 Úprava formátu PDF v GDAL

GeoPDF formát v knihovně GDAL v poslední stabilní verzi 1.11.2[15] využívá knihovnu jenom Poppler nebo PoDoFo. I když je možné kód zkompilovat s oběma knihovnami, při používaní formátu PDF je přednastaven ovladač s knihovnou Poppler a nebo je možné ho změnit pomocí systémové proměnné `GDAL_PDF_LIB`.

Mojí prací bylo přidat podporu použití knihovny PDFium, kterou lze použít samostatně bez externích aplikací (jako v případě PoDoFo, který požaduje nástroj `pdftoppm` z knihovny Poppler). Přítomnost knihovny při kompilaci je vyjádřena existencí maker `HAVE_POPPLER`, `HAVE_PODOFO` a mnou přidané `HAVE_PDFIUM`.

První částí bylo přepsání výběru a použití jedné ze tří dostupných knihoven pomocí bitového pole proměnné `bUseLib` ve zdrojovém souboru `frmts/pdf/pdfdataset.cpp`. Původní systém vybírání pracoval jenom s proměnnou `bUsePoppler` typu bool, tudíž dvou-stavovou hodnotou (pozitivní hodnota (TRUE) znamenala použití knihovny Poppler, při negativní hodnotě (FALSE) se použila knihovna PoDoFo). Způsob nastavení začíná na řádku 2752. V případě existence jenom jednoho z maker se nastavila odpovídající hodnota proměnné `bUsePoppler`, jinak se ověřovala hodnota systémové proměnné `GDAL_PDF_LIB`. Mnou implementovaný systém připravuje možnost využití i dalších PDF knihoven jednodušeji. Na začátku konstruktoru třídy `PDFDataset` se nastaví jednotlivé bity pro každou knihovnu s existencí makra `HAVE_`. Pozice odpovídající knihovny je nastavená v proměnné `bHasLib` pomocí makra `PDFLIB_*` (např. `PDFLIB_PODOFO`), kde makro `PDFLIB_COUNT` má hodnotu o jedno větší než poslední podporovaná knihovna a slouží pro inicializaci proměnné `bHasLib` a `bUseLib`. Díky proměnné typu `bitset` ze standardní C++ knihovny `bitset` je možné rychlým způsobem ověřit existenci alespoň jedné přítomné knihovny pomocí metody `count()`, která vrací počet 1 v bitovém poli. Současně je rychlé určení existence více knihoven a nutnost vybrat jednu z nich, pokud je počet větší než 1. V tomto případě následuje nastavení vybrané knihovny v proměnné `bUseLib` podle názvu knihovny systemové proměnné `GDAL_PDF_LIB`. V závěru je testován počet nastavených knihoven (může být nejvýš jedna) a bitový součin s proměnnou `bHasLib` ověří existenci knihovny. Tímto je ošetřen případ, kdy systémová proměnná má nastavenou hodnotu PDF knihovny, která není použitá při kompilaci GDAL.

Pro zachování přístupu při parsování PDF objektu obecnými metodami třídy `PDFDataset` byla vytvořena třída `GDALPDFObjectPdfium` a další potřebné odvozené třídy v souborech `frmts/pdf/pdfobject.cpp` a `frmts/pdf/pdfobject.h`. Tyto třídy mapují jednotlivé funkce a datové typy používané v knihovně GDAL na funkce a typy knihovny PDFium.

[15] Verze GDAL k datu psaní práce 3.5.2015

Druhou částí je přidání načítání souboru pomocí funkcí `FPDF_LoadDocument()` a ověření požadované stránky pomocí `doc->GetPage( pageNum )`. Jelikož je knihovna PDFium určená především pro jednoduchou rasterizaci PDF souborů, neočekává využití ve vícevláknových aplikacích. Nemožnost rasterizace a čtení dat z jednoho souboru vícevláknově je důsledkem původního tvůrce knihovny (Foxit Software). Bohužel, knihovna využívá globálních proměn pro zjednodušení kódu, který ale znemožňuje použití knihovny i při rasterizaci a čtení různých souborů vícevláknově. Z tohoto důvodu jsem implementoval funkce `LoadPdfiumDocumentPage()` a `UnloadPdfiumDocumentPage()` pro otevírání a zavírání PDF souborů s využitím globální mapy struktury, která udržuje názvy otevřených souborů, seznam otevřených stránek, počet jejich použití a zámky pro čtení souborů. V případě volání GDAL funkce `GDALOpenShared`, která interně otevírá pro každé nové vlákno nový objekt GDALDataset, je při použití knihovny PDFium globálně ověřována cesta otevřeného souboru a při již otevřeném souboru se nevytváří nový objekt knihovny PDFium, ale používá se existující objekt. V praxi je tedy rasterizace a čtení PDF souborů jednovláknová, ale podporuje vícevláknovost knihovny GDAL a nezpůsobuje pády aplikací, které ji využívají. Takové řešení je také šetrné pro paměť, kdy se nemusí udržovat paměťově náročný soubor po dobu běhu programu (jedná se o PDF soubory, které mají obrovské počty vrstev, v paměti může takový objekt PDFium zabírat až 1GB).

Třetí částí je přidání čtení požadovaného obdélníku pixelů ve funkci `ReadPixels()`. Samotné čtení se skládá z těchto fází:

- Získání zámku pro čtení daného souboru a zámku pro použití knihovny PDFium.
- Parsování obsahu a rasterizace v případě prvního přístupu k souboru.
- Příprava bitmapového objektu knihovny PDFium a vyplnění barvou pozadí, která je nastavená na bílou barvu.
- Nastavení rozměru čteného obdélníku podle požadavku z metody `RasterIO()`.
- Samotné čtení pixelů pomocí funkce `FPDF_RenderPageBitmap()`.
- Uvolnění zámku souboru a knihovny.
- Transformace dat z bitmapového objektu typu `B, G, R, Alpha` na typ `R, G, B (, Alpha)`, který je zpracováván knihovnou GDAL.

V poslední části implementace jsem přidal vytvoření abstraktních náhledů pro každý kanál použitý souborem PDF (`GDALRasterBand`). Jedná se o svévolné (arbitrary) náhledy, které mohou být použity pro čtení pixelů, ale nejsou zkoumané knihovnou GDAL při výpočtech histogramu, minimální a maximální hodnoty pixelů a pod. Každý takový náhled má předvypočítaný celkový rozměr, pevný rozměr bloků (dlaždic) a nastavenou úroveň rozlišení. Pomocí úrovně rozlišení se počítá hodnota měřítka použitého ve funkci `ReadPixels()`. Kvůli těmto svévolným náhledům jsem přidal proměnnou `nLastBlockResolution`, na základě které se v objektu `PDFDataset` ukládá poslední blok pro případné opakované čtení.

Všechny změny popsané v této kapitole jsou implementované v samostatné větvi veřejného repozitáře na GitHub[16]. Změny byly implementovány pro hlavní větev (master) a pro větev verze 1.11 (1.11), která je poslední stabilní verzí v době psaní této práce. Změny jsou připravené k odeslání žádosti o přidání do knihovny GDAL.

[16] Větev `pdfium-master` klonu knihovny GDAL https://github.com/klokantech/gdal/tree/pdfium-master

4.4.4 Multiplatformní kompilace GDAL

Přidání podpory knihovny PDFium pro zpracování formátu PDF vyžaduje změny v konfiguraci kompilace knihovny GDAL. Pro operační systémy Linux a Mac OS X jsem upravil soubor `GNUmakefile` přidáním definice makra `HAVE_PDFIUM` a přidání cest k hlavičkovým souborům. Podobná změna je i v souboru `makefile.vc`, kterým se kompiluje GDAL na operačních systémech Windows.

```
+ifeq ($(HAVE_PDFIUM),yes)
+CPPFLAGS +=  -DHAVE_PDFIUM
+endif

-CPPFLAGS := $(GDAL_INCLUDE) $(CPPFLAGS) -I../vrt $(POPPLER_INC) \
  $(PODOFO_INC)
+CPPFLAGS := $(GDAL_INCLUDE) $(CPPFLAGS) -I../vrt $(POPPLER_INC) \
  $(PODOFO_INC) $(PDFIUM_INC)
```

Zdrojový kód 4.5: Ukázka změny v souboru `frmts/pdf/GNUmakefile` pro kompilaci na OS Linux a Mac OS X

```
-EXTRAFLAGS = -I..\vrt $(POPPLER_EXTRAFLAGS) $(PODOFO_EXTRAFLAGS)
+EXTRAFLAGS = -I..\vrt $(POPPLER_EXTRAFLAGS) $(PODOFO_EXTRAFLAGS) \
  $(PDFIUM_EXTRAFLAGS)

+!IFDEF PDFIUM_ENABLED
+PDFIUM_EXTRAFLAGS = $(PDFIUM_CFLAGS) -DHAVE_PDFIUM
+!ENDIF
```

Zdrojový kód 4.6: Ukázka změny v souboru `frmts/pdf/makefile.vc` pro kompilacií na Windows OS

Do souboru `configure.in` jsem přidal detekci knihovny PDFium pomocí argumentu `--with-pdfium[=path]`. Cesta ke knihovně vychází ze standardního způsobu instalace knihoven v prostředí operačního systému Linux, tehdy cesta vyžaduje existenci složek `lib/` a `include/`. Pro nalezení knihovny je použit testovací soubor, který se kompiluje s nastavenou cestou k PDFium. Po úspěšné kompilaci testovéhou souboru se nastaví proměnné `HAVE_PDFIUM`, `PDFIUM_INC` a `PDFIUM_PLUGIN_LIB`.

Pro operační systémy Windows OS jsem přidal do souboru `nmake.opt` zakomentované řádky, které popisují způsob přidání knihovny PDFium do GDAL. Nastavují se proměnné `PDFIUM_ENABLED`, `PDFIUM_CFLAGS` a `PDFIUM_LIBS`. V případě Windows OS je potřebné do proměnné `PDFIUM_LIBS` přidat všechny statické knihovny, které byly vytvořené při kompilaci knihovny PDFium, viz sekce 4.4.2.

Pro všechny operační systémy jsem přidal podporu pro vytvoření zásuvného modulu `gdal_PDF.$ext`[17], změnami v souborech `frmts/pdf/GNUmakefile` a `frmts/pdf/makefile.vc`. Doporučená konfigurace a příkaz ke kompilaci pro operační systém Linux a Mac OS X je: `$ ./configure --without-libtool --with-python --with-threads --with-pdfium` `$ cd frmts/pdf/ ; make plugin`.

[17] Přípona zásuvného modulu pro Windows je `dll`, Linux `so` a Mac OS X `dylib`.

```
+#PDFIUM_ENABLED = YES
+#PDFIUM_CFLAGS = -Ie:/pdfium-svn/
+#PDFIUM_LIB_DIR = e:/pdfium-svn/build/Release/lib
+#PDFIUM_LIBS = $(PDFIUM_LIB_DIR)/pdfium.lib $(PDFIUM_LIB_DIR)/bigint.lib \
+#  $(PDFIUM_LIB_DIR)/fdrm.lib $(PDFIUM_LIB_DIR)/formfiller.lib \
+#  $(PDFIUM_LIB_DIR)/fpdfapi.lib $(PDFIUM_LIB_DIR)/fpdfdoc.lib \
+#  $(PDFIUM_LIB_DIR)/fpdftext.lib $(PDFIUM_LIB_DIR)/freetype.lib \
+#  $(PDFIUM_LIB_DIR)/fxcodec.lib $(PDFIUM_LIB_DIR)/fxcrt.lib \
+#  $(PDFIUM_LIB_DIR)/fxedit.lib $(PDFIUM_LIB_DIR)/fxge.lib \
+#  $(PDFIUM_LIB_DIR)/pdfwindow.lib
```

Zdrojový kód 4.7: Ukázka změny v souboru `nmake.opt` pro kompilaci na Windows OS

4.5 Integrace zpracování do serveru IIPImage

Podkapitola 3.3 se věnuje projektu IIPImage. Jeho serverová část využívá pro čtení obrazových souborů knihovnu LibTIFF. V této podkapitole popisuji implementační detaily integrace serverové části QtGDAL do projektu IIPImage.

4.5.1 Úprava zdrojového kódu

Pro práci se soubory pomocí knihovny GDAL jsem přidal novou třídu `GdalImage`, která dědí od třídy `IIPImage`. Třídu jsem implementoval podobným způsobem jako je implementovaná třída `KakaduImage`. Třída obsahuje mimo jiné následující funkce, které jsem přidal nebo upravil:

1. `InitialiseLibrary()` - statická veřejná funkce,
2. `IsFileSupported(std::string)` - statická veřejná funkce,
3. `needOverviews()` - privátní metoda,
4. `process(res, levels, x, y, w, h, d)` - privátní metoda, přebraná z třídy `GdalImage` a upravená pro GDAL.

První základní veřejnou funkcí je `IsFileSupported()`, která ověří zda může být soubor otevřen knihovnou GDAL. Tato funkce při prvním volání inicializuje knihovnu GDAL pomocí funkce `InitialiseLibrary()`.

```
+  GDALDataset * ds = (GDALDataset *) GDALOpen(path.c_str(), GA_ReadOnly);
+  if(ds == NULL)
+    return 0;
+  GDALClose(ds);
+  return 1;
```

Zdrojový kód 4.8: Ukázka funkce `GdalImage::IsFileSupported()` implementovaná v souboru `src/GdalImage.cc`.

Inicializační funkce je implementovaná v souboru `GdalInit.cc`. Pro operační systémy Windows a Mac OS X nastaví cesty k datovým souborům a zásuvným modulům knihovny GDAL. Součástí inicializace je registrace všech podporovaných rastrových formátu a nastavení potlačování chyb z GDAL.

Při otevírání souboru zděděné funkce `openImage()` se načítají informace o obrazu a ověří se možnost rychlého zpracování souboru. Rychlost zpracování je ověřovaná funkcí `needOverviews()`, která je součástí serverové části aplikace QtGDAL a je popsaná v kapitole 5. V případě vyhodnocení pomalého zpracování je vyvolaná výjimka s textem, že soubor potřebuje náhledy. Při načítání informací o obrazu jsou vytvořené virtuální úrovně rozlišení, které umožňují generovat dotazy pro všechny úrovně přiblížení od jedné dlaždice po nativní rozlišení obrazu. Počet těchto virtuálních rozlišení je zapsán v proměnné `numResolutions`.

Implementace funkcí `getTile()` a `getRegion()` jsou přebrané z třídy `GdalImage`, protože se jedná o stejný způsob práce nad obrazem. Obě funkce volají na závěr privátní metodu `process()`, která na základě parametrů načte část obrazu do poskytnuté mezipaměti, která je následně odeslána serverem klientovi jako odpověď na žádost.

Metoda přímým způsobem zapisuje data do mezipaměti z obrazu podle nastavených parametrů:

- `pixelSpace` - mezera mezi pixelama, v našem případě to je počet kanálů obrazu (3 pro RGB),
- `lineSpace` - mezera mezi řádky obdélníku, odpovídá šířce žádané oblasti a násobku počtu kanálů,
- `bandSpace` - mezera mezi kanály obrazu, pro IIPImage to je počet bytů typu dat (pro 16-bitový typ to jsou 2 Byty).

Hodnoty pozice a velikosti oblasti jsou vypočítané jako bitový posun doleva originální hodnoty o hodnotu rozdílu mezi počtem všech rozlišení (`numResolutions`) a žádanou úrovní rozlišení.

Integrace do IIPImage vyžadovala změny stávajících souborů, zejména detekce typu žádaného souboru v metodě `testImageType()` třídy `IIPImage`. Jednoduchou detekci souborů typu TIFF (podle hlavičky souboru) jsem nahradil přidáním veřejné funkce `IsFileSupported()` do třídy `TPTImage`. Tato třída pomocí knihovny LibTIFF pracuje velmi rychle se vstupním souborem, který musí být složený z dlaždic a sestavený do pyramidy s více rozlišením (viz obrázek 3.1 v podkapitole 3.3 na straně 18), tzv. *Tiled Pyramidal Tiff Images*. Funkce `IsFileSupported()` nejdříve otevře soubor pomocí knihovny a pak zjistí velikost dlaždice v souboru. Pokud vstupní obraz nemá dlaždice, pak velikost bude nulová, a v tomto případě není vstupní soubor podporován touto třídou. Takový soubor nicméně může otevřít a zobrazit právě implementovaná třída `GdalImage`.

```
+  if( ( tiff = TIFFOpen( path.c_str(), "r" ) ) == NULL )
+    return 0;
+  unsigned int tile_width = 0, tile_height = 0;
+  TIFFGetField( tiff, TIFFTAG_TILEWIDTH, &tile_width );
+  TIFFGetField( tiff, TIFFTAG_TILELENGTH, &tile_height );
+  TIFFClose( tiff );
+  // Insist on a tiled image
+  return !((tile_width == 0) && (tile_height == 0));
```

Zdrojový kód 4.9: Ukázka funkce `TPTImage::IsFileSupported()` přidaná v souboru `src/TPTImage.cc`.

Všechny operace vyžadující knihovnu GDAL jsou chráněné makrem `HAVE_GDAL`, které je při konfiguraci nastaveno v případě nastavení cesty ke knihovně GDAL. Upravený kód projektu je proto možné používat i bez knihovny GDAL.

Výše popsané změny jsem implementoval v rámci samostatné větve `gdal-thesis` na serveru GitHub firemního klonu projektu IIPImage[18].

4.5.2 Úprava konfigurace pro kompilování

Do konfiguračního souboru projektu jsem přidal zpracování proměnné `--with-gdal=/path/`, která je ověřená existencí hlavičkového souboru knihovny `gdal.h`. Správné nastavení cesty přidá do konfigurace projektu následující změny:

- makro `HAVE_GDAL`,
- cesta k hlavičkovým souborům knihovny `INCLUDES += -I$gdal_path/include/gdal`,
- objekty pro kompilování `GdalImage.o GdalInit.o`,
- linkování knihovny GDAL `-L$gdal_path/lib/ -lgdal`.

V případě nainstalování vývojářského balíčku knihovny `libgdal-dev` do standardní systémové cesty (`/usr/lib/` a `/usr/include/`) je příkaz konfigurace s knihovnou GDAL:
`$ ./configure --with-gdal=/usr`

Postup kompilace serveru je dostupný na stránkách projektu http://iipimage.sourceforge.net/documentation/server/.

[18] Integrace GDAL do IIPImage projektu https://github.com/klokantech/iiifserver/tree/gdal-thesis

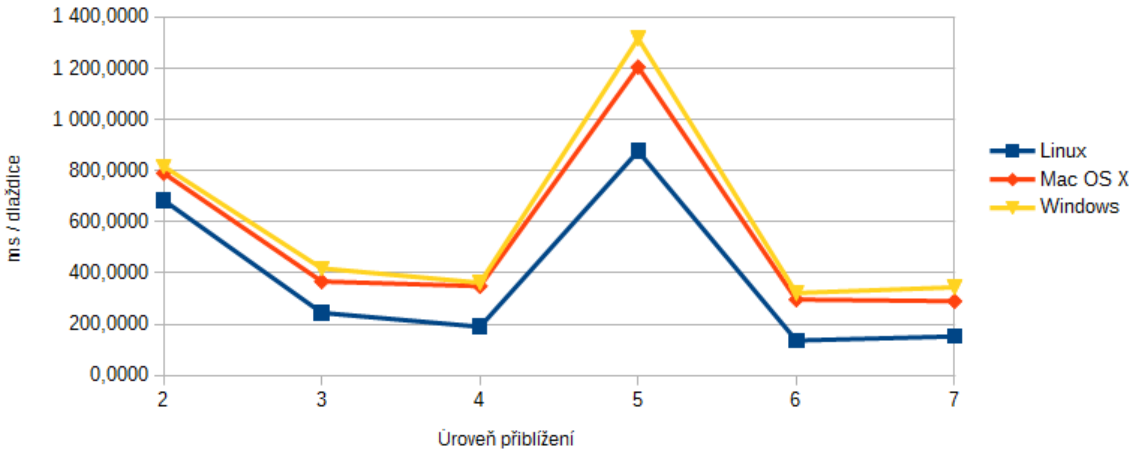
Kapitola 5

Optimalizace rychlosti zpracování požadovaných dlaždic obrazu

Pro semestrální projekt jsem připravil rychlostní testy zpracování různých souborů a testoval jsem je na třech rozličných hardwarových sestavách s jiným operačním systémem. Z průběhu testování a výsledků testů jsem usoudil, že operační systém nemá přímý vliv na rychlost zpracování, ale hardwarová sestava a typ obrazu ovlivňují zpracování v těchto bodech:

- CPU - počet pracujících vláken a frekvence jádra,
- disk - typ (SSD nebo HDD) a rychlost disku,
- typ souboru - možnost čtení náhodných pozic (např. prostředek souboru),
- komprese obrazových dat.

Na obrázku 5.1 je graf závislostí průměrného času zpracování jedné dlaždice pro testovací soubor s rozlišením 21 600 x 10 800 px (233 MPx), typu PNG. Operační systém Linux pracoval s obrázkem uloženým na SSD disku, přičemž Mac OS X měl k dispozicí HDD s rychlostí 7200 otáček za minutu. Operační systém Windows byl testovaný na HDD disku s 5400rpm.

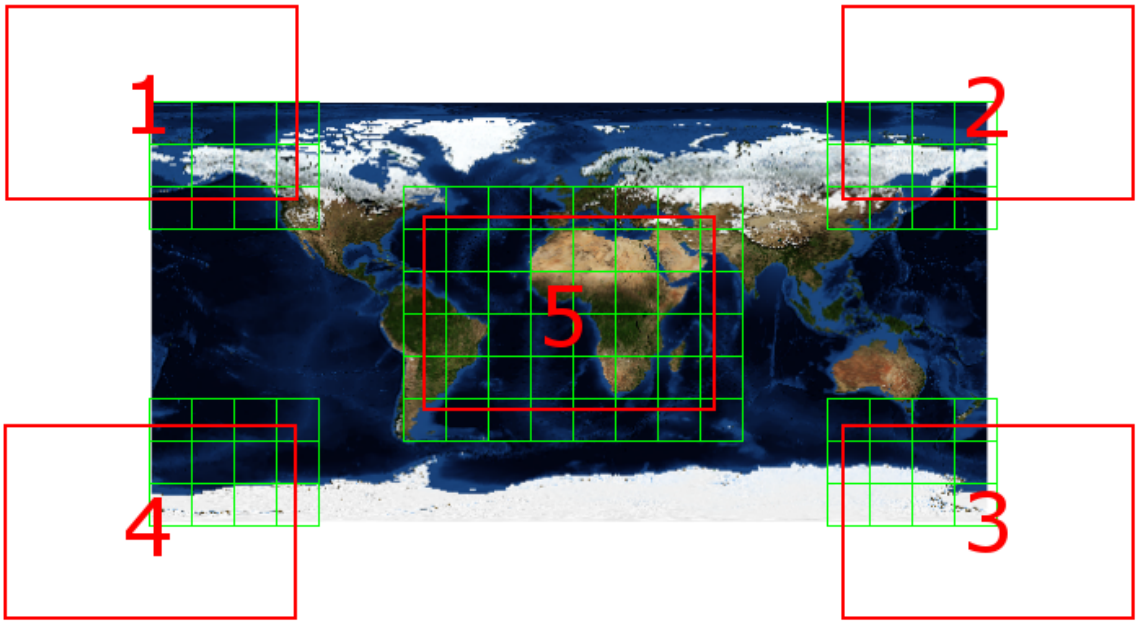


Obrázek 5.1: Průměrný čas načítání dlaždice pro úroveň přiblížení

Základní způsob optimalizace je generování náhledů, které je časově závislé na velikosti a typu vstupního obrazu. V první fázi optimalizace aplikace generovala náhledy pro všechny úrovně přiblížení, které vedlo k většímu pomocnému souboru a delšímu času otevíraní souboru, při kterém se náhledy generovaly. V následujících částech zprávy popisuji výsledky testů měření aplikace bez optimalizace a pak s optimalizací. V poslední části jsou implementační detaily, které využívají výsledky z těchto testů.

5.1 Načítání nativního obrázku

Při otevření souboru aplikace nejdříve zobrazí celý obraz uprostřed zobrazovací plochy. Úroveň přiblížení je 0, tudíž vstupní obrázek se zobrazí jako jedna dlaždice. Pokud je dlaždice zaplněna na méně než 75% největší strany vstupního obrazu, pak se nastaví výchozí úroveň přiblížení na 1, kdy jsou požadované nejvíce 4 dlaždice. Pro nativní obrázek běžného formátu[1] to znamená čtení celého souboru a následné zmenšení načtené oblasti.



Obrázek 5.2: Způsob testování aplikace - pozice zobrazovací plochy.

Test rychlosti aplikace je zobrazen na obrázku 5.2. Pro každou testovanou úroveň přiblížení je zobrazovací plocha postupně přesunuta na místa od 1. po 5. (střed obrazu). Pro první dvě úrovně jsou testované plochy vrchního a spodního středu vstupního rastru[2]. Zelené čtverce představují dlaždice, které jsou během testování zpracovávané. Testování bylo vyhodnoceno na stejném obrazovém vstupu, který byl pro každé rozlišení vytvořen zmenšením původního vstupu o rozměrech 86400 x 43200 pixelů. Testoval jsem nejpoužívanější formáty rastrů, ve kterých se rozměrná rastrová data ukládají. Netestoval jsem formáty, které jsou speciálně určené pro vysoké rozlišení (typu JPEG 2000, ECW[3] nebo MrSID[4]),

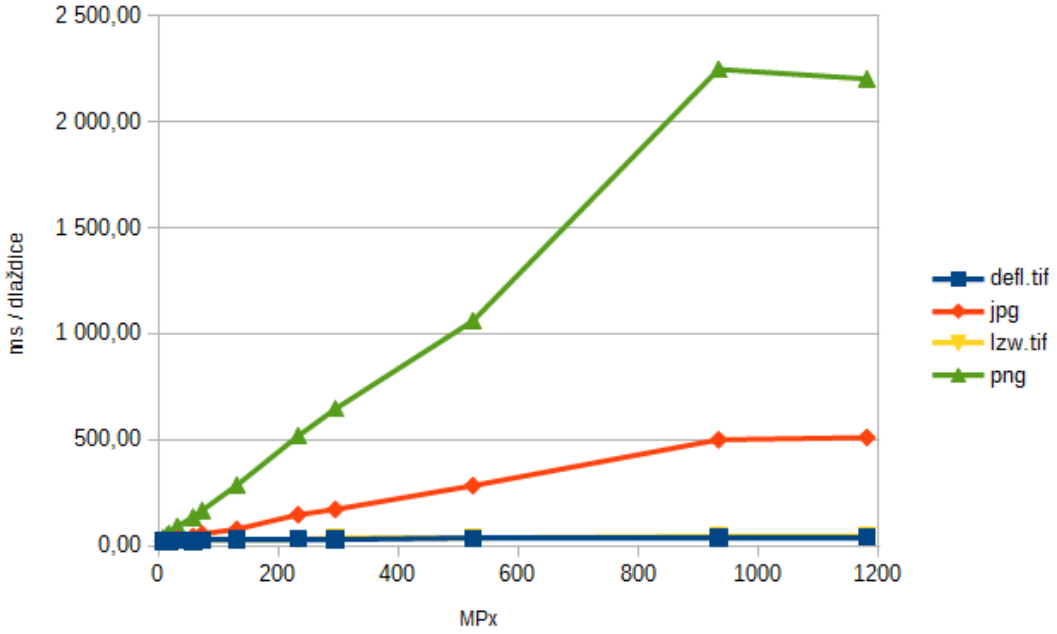
[1] Soubor bez pyramidy dlaždic, viz obrázek 3.1 v podkapitole 3.3 na straně 18

[2] Zobrazovací plocha je pro vrchní střed umístěna uprostřed spojnice 1., 2. a 5. testovacího vstupu z obrázku 5.2

[3] Proprietární rastrový formát ERDAS ECW firmy Hexagon Geospatial http://hexagongeospatial.com/products/data-management-compression/ecw

[4] Proprietární formát MrSID firmy LizardTech http://www.lizardtech.com/products/lidar/

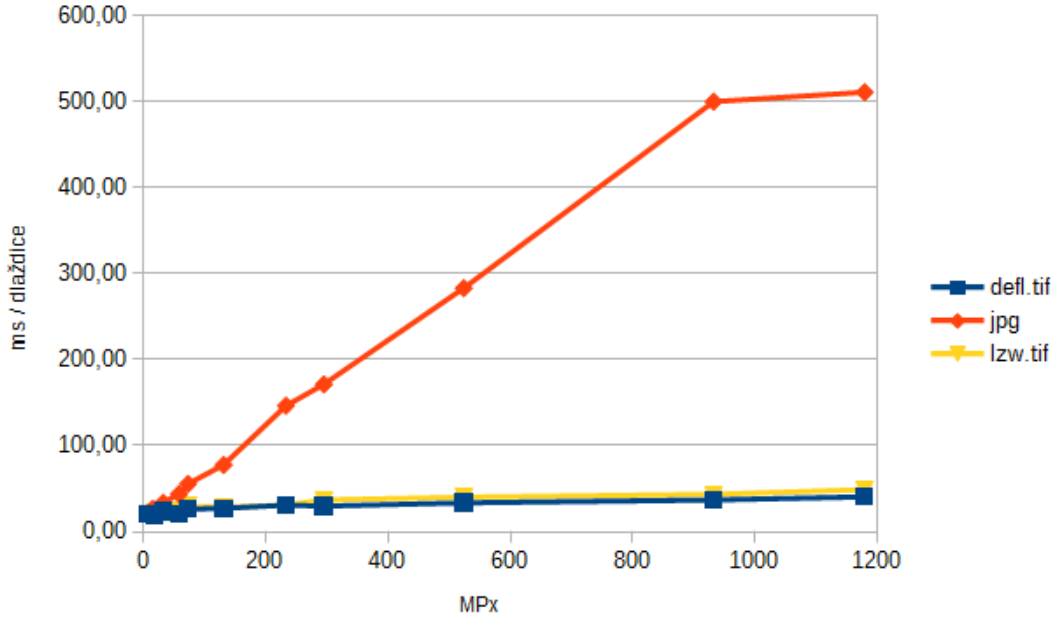
protože jejich výkonnost je dostatečná i pro vysoké rozlišení. Testování probíhalo na disku SSD, procesor Intel i7-3770 s frekvencí 3.2 GHz s omezením na 6 vláken CPU (z 8 možných).



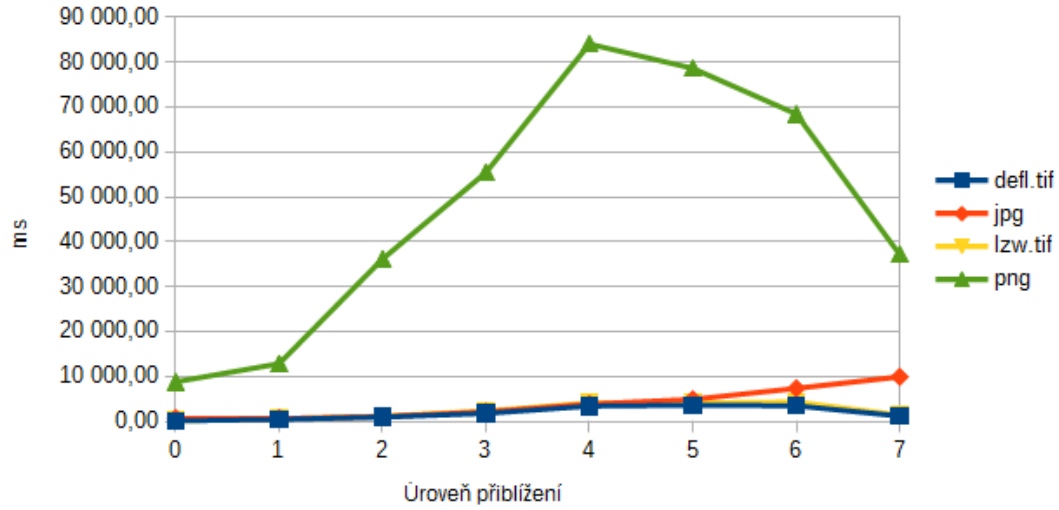
Obrázek 5.3: Časová závislost načítání dlaždice na rozlišení.

Na obrázku 5.3 je graf průměrného času načítání jedné dlaždice v závislosti na rozlišení vstupního obrazu. Graf zobrazuje časy zpracování nativní úrovně přiblížení (nejvyšší pro vstupní soubor), kdy se načítá dlaždice jako část obrazu, která není zmenšená. Stejný průběh mají i grafy pro každou úroveň přiblížení, proto je neuvádím. Z grafu jasně plyne, že formát **PNG** má nejpomalejší zpracování, které lineárně roste rychleji než formát **JPG**. Formát **TIFF** s kompresí **Deflate** (defl) a **LZW** má dostatečnou rychlost zpracování, kdy pro nejvyšší testované rozlišení *1 180 MPx* průměrný čas načítání jedné dlaždice nepřesáhne 50 ms. Obrázek 5.4 přibližuje poměr časů pro zpracování formátu **TIFF** (s oběma typy komprese) a **JPG**. Z těchto grafů jsem určil hodnoty, které by měly vést k rychlejšímu zpracování, které je popsané v poslední části této kapitoly. Přiblížení poměru do 200 MPx je v příloze C.1.1 zobrazeno na grafu C.2 na straně 61. Obrázek 5.3 obsahuje předposlední rozlišení, které se vychyluje od předpokladané přímky. Důvodem je menší rozdíl v počtu dlaždic než počet vláken, které současně načítávají vstupní soubor. V obou případech testy proběhly za téměř stejnou dobu, kdy v případě menšího rozlišení (933 MPx) v posledním kole 4 vlákna nepracovala[5].

[5] Větší obrázek má 54 dlaždic a menší 50 dlaždic. $\lceil 54/6 \rceil = 9 = \lceil 50/6 \rceil$.



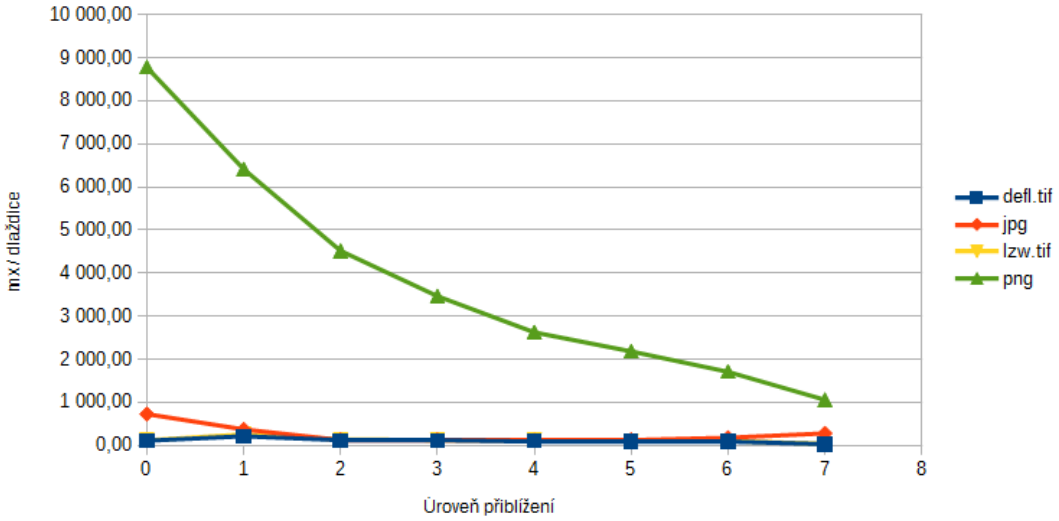
Obrázek 5.4: Časová závislost načítání dlaždice na rozlišení (mimo formát PNG).



Obrázek 5.5: Časová závislost načítání vstupního obrazu velikosti 524 MPx pro každou úroveň přiblížení.

Graf 5.5 vyjadřuje průměrný čas zpracování celého vstupního souboru. Nativní úroveň přiblížení[6] (7) je nejrychlejší pro zpracování jedné dlaždice, protože ostatní úrovně zmenšují dlaždice podle požadované velikosti. Časově nejkritičtější úroveň přiblížení je 4., kde 32 dlaždic má 8-krát zmenšené obě strany, tudíž má 64-krát méně pixelů. Přepočítáním času zpracování na jednu dlaždici dostaneme očekávaný výsledek - čas načítání dlaždice stoupá s úměrně se zmenšující úrovní přiblížení (viz obrázek 5.6). Grafy 5.5 a 5.6 jsou výsledkem testování souboru s rozlišením 32 400 x 16 200 (524 MPx) uloženém na SSD disku a s využitím 6 pracujících vláken CPU.

[6]Největší úroveň charakterizuje nativní rozlišení obrazu



Obrázek 5.6: Časová závislost načítání jedné dlaždice vstupního obrazu velikosti 524 MPx pro každou úroveň přiblížení.

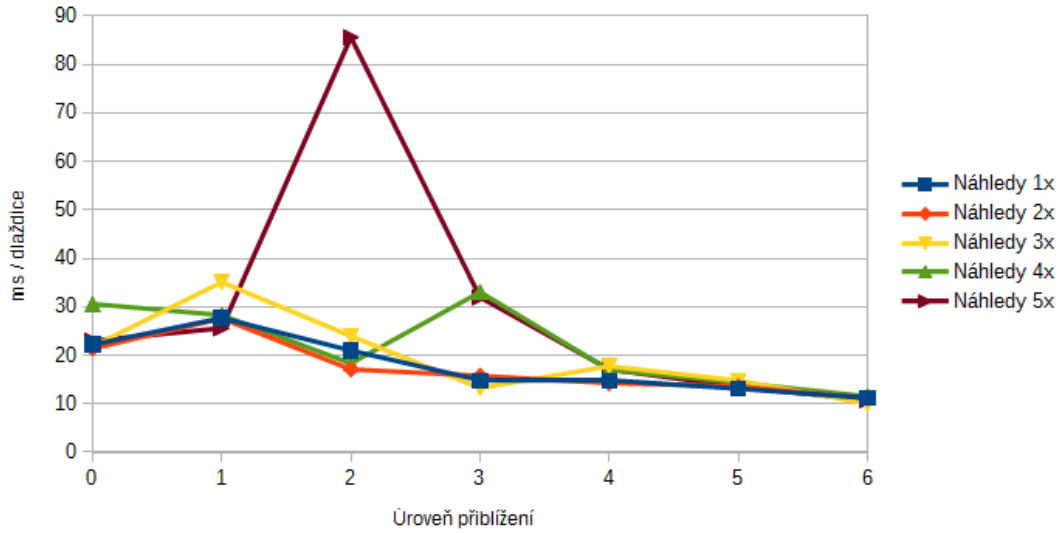
5.2 Generování náhledů vstupního obrazu

Při otevírání souboru je možné vytvořit náhledy - pyramida zmenšených obrazů. Obraz náhledu má zmenšenou x-krát každou stranu, kde číslo x je násobkem čísla 2. Hlavní výhodou těchto náhledů je zrychlení čtení obrazu pro menší než nativní úroveň přiblížení za cenu vytvoření menšího dodatečného souboru. Jedinou nevýhodou je sériové zpracování - není možné využít výhodu více vláken CPU. V první fázi optimalizace aplikace generovala náhledy pro každou úroveň přiblížení. Náhledy jsou ve formátu TIFF s kompresní metodou *LZW*. Metoda převzorkování použitá při generování náhledu je *average* - průměrná hodnota okolí bodu. Nejdříve jsem se rozhodl otestovat rychlost zpracování, pokud se náhledy budou generovat pro každou i-tou úroveň přiblížení. Pak jsem otestoval rychlost všech vstupních testovacích souborů s různým rozlišením pro vybraný způsob generování náhledů.

První část testu jsem provedl pro testovací soubor s rozlišením 524 MPx. Obrázek 5.7 zobrazuje časovou závislost zpracování jedné dlaždice vzhledem k požadované úrovni přiblížení. Legenda *Náhledy Ix* znamená, že se náhledy vytvářely pro každou I-tou úroveň, počínajíc od největší (nativní) úrovně přiblížení. Vstupní testovací soubor má rozlišení 32 400 x 16 200 px, proto má osm úrovní, 0 až 7. Poslední (7.) úroveň je nativní obraz, pro který se náhledy nevytvářejí. Pro rychlejší orientaci uvádím výčet úrovní vytvořených náhledů:

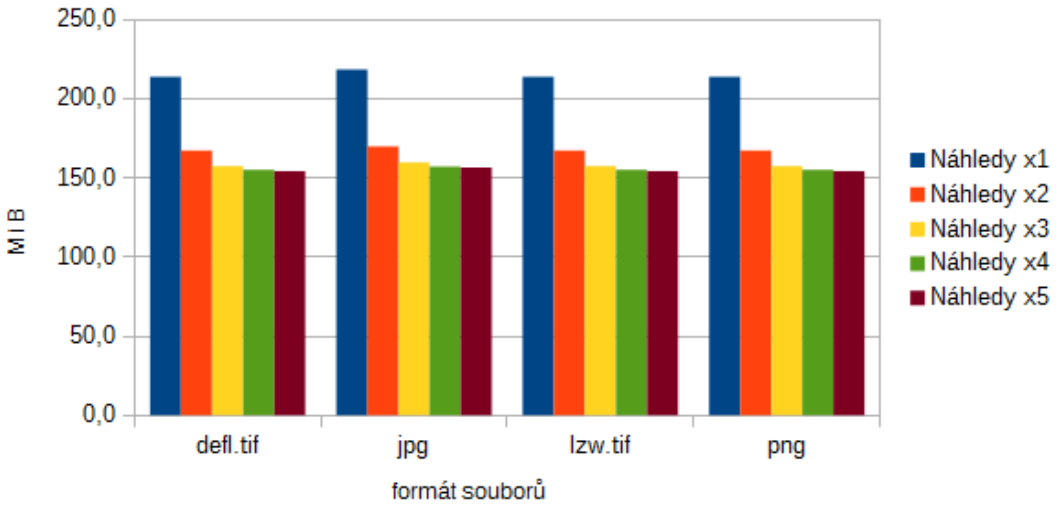
- *Náhledy 1x* - 6, 5, 4, 3, 2, 1, 0;
- *Náhledy 2x* - 6, 4, 2, 0;
- *Náhledy 3x* - 6, 3, 0;
- *Náhledy 4x* - 6, 2;
- *Náhledy 5x* - 6, 1.

Graf 5.8 vyjadřuje poměry velikosti pomocných souborů, ve kterých se vytvářejí náhledy. Z grafu je patrné, že snížení počtu náhledů neovlivňuje výrazně velikost pomocného souboru. Proto toto kritérium není velmi důležité při rozhodování o zvoleném způsobu generování

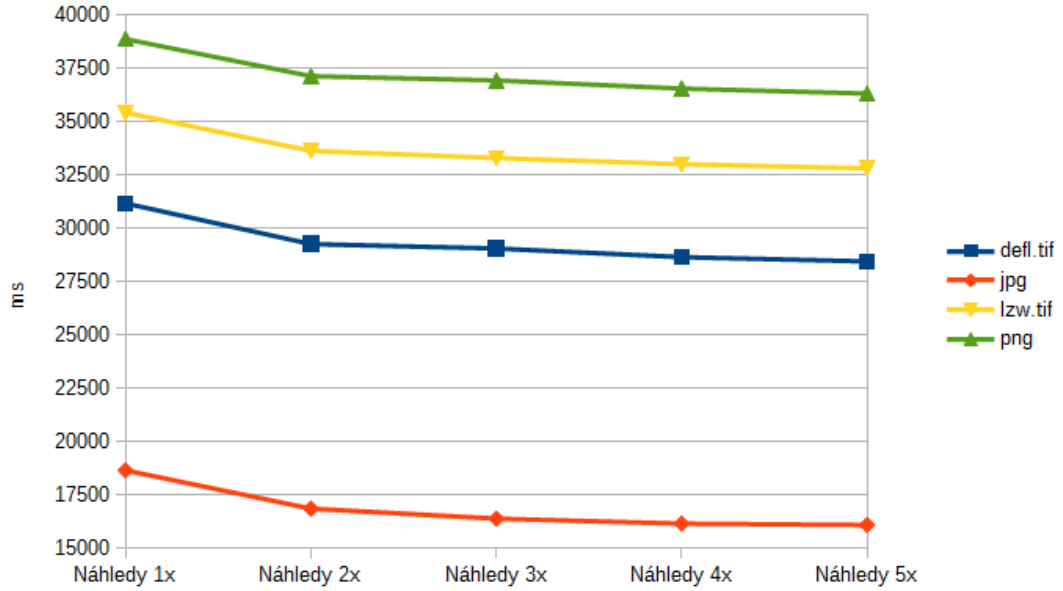


Obrázek 5.7: Časová závislost načítání jedné dlaždice vstupního obrazu pro každou úroveň přiblížení s vytvořenými náhledy. Vstupní soubor má rozlišení 524 MPx.

jednotlivých stupňů náhledů. Nicméně podle výše zmíněného obrázku 5.7 je vhodné zvolit generování každé 2. nebo 3. úrovně přiblížení. Pokud se generuje každá 5. úroveň, tak pro úroveň přiblížení 2 je nejvyšší čas zpracování dlaždice, protože se zmenšuje s nejbližší větší úrovně, v tomto případě to je úroveň 6. (viz výčet úrovní výše). Podobně to je pro úroveň přiblížení 3 při generování každé 4. nebo 5. úrovně. Časový rozdíl mezi zpracováním jedné dlaždice při generování každé 1., 2. a 3. úrovně není kritický. Z grafu 5.9 vyplývá, že vytváření náhledů má stejný časový rozdíl mezi jednotlivými způsoby, který je nezávislý na formátu vstupního souboru. Čas generování náhledů je přímo závislý na formátu vstupního obrazu. Grafy 5.7 a 5.8 jsou nezávislé na formátu vstupního souboru, protože formát generovaných náhledů je stejný (TIFF s LZW kompresí). Poměr velikosti vstupního obrazu k vytvořeným pomocným souborem je zobrazen v grafu C.3 v příloze C.1.2 na straně 61



Obrázek 5.8: Poměr velikostí pomocných souborů s vytvořenými náhledy pro vstupní soubor s rozlišením 524 MPx.

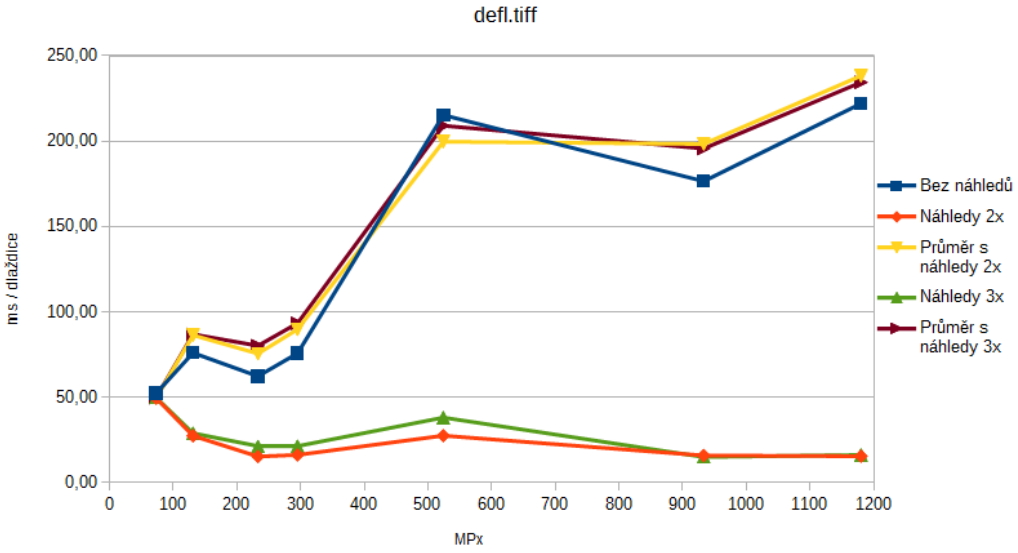


Obrázek 5.9: Časová závislost vytváření náhledů více způsoby pro vstupní soubor s rozlišením 524 MPx.

Pro druhou část testu jsem vybral způsob generování každé druhé a třetí úrovně náhledu (z legendy předchozích grafů *Náhledy 2x* a *Náhledy 3x*). Každý vstupní testovací soubor byl testován s oběma způsoby generování náhledů. Výsledný graf 5.10 znázorňuje časový průběh generování jedné dlaždice pro různé rozlišení, kde formát vstupních obrazů byl **TIFF** s kompresí typu *Deflate*. Hodnoty grafu **Průměr s náhledy** *I***x** jsou výsledkem součtu naměřených časů načítání jedné dlaždice s náhledy vytvořenými pro každou *I*-tou úroveň (viz výše popsané a vysvětlené způsoby generování náhledů) a časem vytváření náhledů při otevírání souboru, který je přepočítaný na počet všech dlaždic, které byly žádané při testování. Tento údaj by měl reprezentovat průměrný čas přístupu k získání jedné dlaždice. Graf pro formát vstupních souborů TIFF má časy zpracování jedné dlaždice menší s generovanými náhledy než bez náhledů, nicméně pokud připočítáme čas generování náhledů (kdy uživatel musí čekat), tak se čas zpracování vyrovnává časům bez náhledů. Tento testovací graf vyjadřuje skutečnost, že optimalizace zapnutím generování náhledů pro formát typu TIFF není velmi efektivní, tudíž tvorba náhledů není potřebná. Opačným případem je graf 5.11 zpracování jedné dlaždice vstupních souborů formátu **PNG**. Z grafu jasně plyne skutečnost, že i průměrný čas načítání jedné dlaždice s časem pro generování náhledů má výrazně rychlejší optimalizaci. Formát vstupních souboru **JPG** má taktéž účinnou optimalizaci na rychlost čtení jedné dlaždice, kdy čas testovaných souborů nebyl vyšší než 200ms. Graf časů je na obrázku C.4 v příloze C.1.2 na straně 62.

5.3 Využití výsledků pro optimalizaci aplikace

V předchozích podkapitolách jsem testoval možné způsoby optimalizace aplikace a interpretoval jsem jejich výsledky. Na základě těchto testů jsem optimalizoval aplikaci. Optimalizace spočívá v dynamickém rozhodování o potřebě vytváření náhledů a způsob jejích generování. Společným prvkem rozhodování je formát vstupního souboru, typ komprese a rozlišení vstupního souboru vyjádřené v Mega Pixelech (MPx = 1 000 000 Px). Tato optimalizace je implementovaná v metodách `needOverviews()` a `rebuildOverviews()` třídy



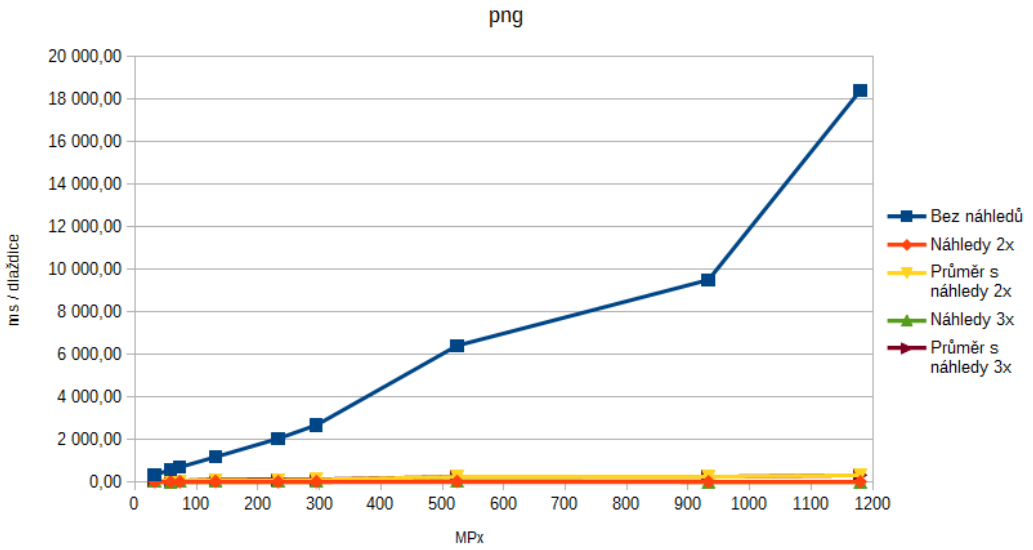
Obrázek 5.10: Časová závislost zpracování dlaždice obrazu (s využitím náhledů) na rozlišení vstupního obrazu (formátu TIFF s kompresí Deflate).

`QGFile` (zdrojový soubor `src/file.cpp`).

Funkce `needOverviews()` rozhoduje, je-li potřeba generovat náhledy pro otevřený soubor. Nejdříve si aplikace připraví proměnné pro rozlišení `MPx`, formát souboru `ftype` a typ komprese `compr`. První podmínka odmítá potřebu náhledů, pokud je rozlišení obrazu menší než 30 MPx. Hodnotu jsem vybral z grafu C.1 (nachází se v příloze C.1.1 na straně 60), která přibližně odpovídá maximální době zpracování jedné dlaždice 100 ms. Druhá podmínka ošetřuje formát vstupního obrazu **TIFF**. Načítání dlaždic z takového typu vstupu je dostatečně rychlé, když není použita žádn8 kompresní metoda. Takový výsledek ale platí jenom když se pracuje s dlaždicemi v nativním rozlišení obrazu. Pokud aplikace potřebuje zpracovat dlaždice v menší úrovni přiblížení, tak čas pro načítání oblasti dlaždice a její zmenšení již přesahuje hranicí 100 ms (viz obrázek 5.12). Z grafu jsem určil hodnotu 500 MPx jako hraniční velikost vstupního obrazu, kdy je při použití **JPG** a nebo žádné kompresní metody potřeba vytvářet náhledy. Pro jiné kompresní metody je hraniční hodnota 400 MPx, vycházejíc z časového průběhu testovaných souborů TIFF s **LZW** kompres9.

Ovladač formátu **JPEG** knihovny GDAL má připravené virtuální náhledy pro každý soubor, který ovladač otevírá. Při testech se takové náhledy neprojevují časově optimalizované, protože se ve většině případů jedná o náhledy s velikost9 čteného bloku *šířka náhledu* * 1. Externí generované náhledy typu TIFF (jak je zmíněno na začátku podkapitoly 5.2) používají bloky o velikosti 256 * 256. Z grafů výše uvedených jsem vybral hodnotu 300 MPx pro potřebu generování náhledů. Podmínka k vytváření náhledů je navíc ošetřená velikost bloku pro již existující náhledy (viz zdrojový kód 5.1). Velikost bloku musí byt stejná pro šířku i výšku.

Poslední podmínkou je samotná existence náhledů. Pokud vstupní soubor má náhledy, tak není potřeba je znovu generovat. Touto podmínkou aplikace ošetřuje nejen vícenásobné otevírání souborů, ale také zrychluje práci s formáty, které již mají náhledy (příkladem jsou speciální formáty pro vysoké rozlišení: MrSID, ECW nebo JPEG2000).

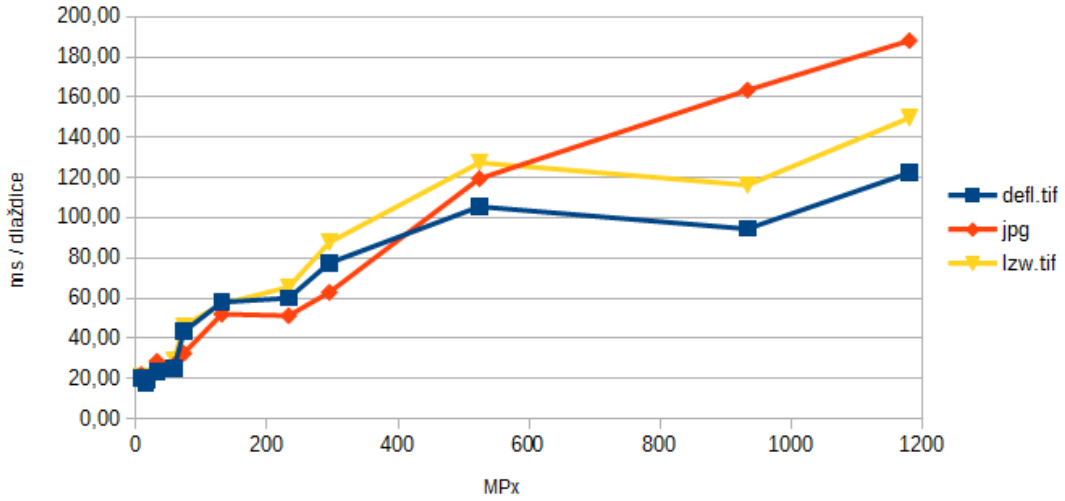


Obrázek 5.11: Časová závislost zpracování dlaždice obrazu (s využitím náhledů) na rozlišení vstupního obrazu (formátu PNG).

```
if(strcmp(ftype, "JPEG") == 0) {
  int ovc = band->GetOverviewCount();
  if(ovc) {
    int nBlockXSize, nBlockYSize;
    GDALRasterBand *overview = band->GetOverview(0);
    overview->GetBlockSize(&nBlockXSize, &nBlockYSize);
    if(nBlockXSize == nBlockYSize)
      return false;
  }
  return (MPx > 300);
}
```

Zdrojový kód 5.1: Ukázka využití testů pro rychlostní optimalizaci aplikace QtGDAL. Dynamické generování náhledů.

Výsledky testů jsem také využil při úpravě zdrojového kódu funkce `rebuildOverviews()` (viz zdrojový kód 5.2), která generuje náhledy pro otevřený soubor, pokud to metoda `needOverviews()` povolí a nebo je uživatelem vynucené generování náhledů. Na základě grafu 5.7 jsem se rozhodl, že aplikace bude generovat náhledy pro každou třetí úroveň přiblížení. Speciální situace nastává, když je počet úrovní přiblížení náhledů násobkem čísla 3. Pokud například klient požaduje dlaždice 1. úrovně přiblížení, pak aplikace načítá oblasti dlaždice z náhledů pro 3. úroveň přiblížení (nejbližší vyšší úroveň, která je násobkem čísla 3) a musí obraz zmenšovat 4-násobně v obou stranách oblasti. Delší čas zpracování se projevuje v grafu 5.7. Z grafu je také vidět, že čas zpracování jediné dlaždice na 0-té úrovni přiblížení, která vychází z náhledů o jednu vyšší úroveň, je menší (viz hodnoty *Náhledy 4x*). Z tohoto důvodu je podmíněné nastavení 1. úrovně přiblížení, pokud by měla být nastavená 0. úroveň přiblížení.



Obrázek 5.12: Časová závislost načítání dlaždice na rozlišení, 4. úroveň přiblížení.

```
// This is maximum zoom levels
// Real number of required overviews are count
int * levels = (int*) CPLMalloc(_maxZoom * sizeof(int));
int count = 0;
// Dynamic overviews
for(int i=1; i <= _maxZoom; i += 3) {
  levels[count] = 1 << i;
  ++count;
}
// Move zoom 0 to zoom 1 - faster performance
// Scale for zoom 0 is 1 << _maxZoom (2 ^ _maxZoom)
if(levels[count-1] == (1 << _maxZoom))
  levels[count-1] >>= 1;
```

Zdrojový kód 5.2: Ukázka využití testů pro rychlostní optimalizaci aplikace QtGDAL. Dynamické generování náhledů.

Obsahem následující kapitoly je také experimentální testování těchto optimalizací na jiných vstupech s využitím různých typů hardwaru a operačních systémů.

Kapitola 6

Experimentální testování a porovnání výsledků práce

Kapitola popisuje experimentální testování různých rozměrných rastrů s využitím rozličných typů hardwaru. Testovací data jsem získal z volně dostupných serverů[1] [2] a z firmy Klokan Technologies GmbH[3]. Seznam základních údajů o testovacích souborech jsou v tabulce 6.1.

V první podkapitole vyhodnocují závislost hardwarových sestav na běhu aplikace. Následující podkapitola porovnává rychlost aplikace QtGDAL s existujícími aplikacemi, které mají stejný cíl - zpřístupnit rozměrné rastrové data.

MPx	**Rozlišení**	**Barevný model**
64	7 800 x 8 300	RGB
88	8 331 x 10 664	Paleta RGB
95	10 699 x 8 943	RGB
134	16 384 x 8 192	RGB
144	12 000 x 12 000	RGB
151	11 049 x 13 716	RGB
161	14 791 x 10 942	Stupně šedi
233	21 600 x 10 800	RGB
423	29 566 x 14 321	RGB
466	21 600 x 21 600	RGB

Tabulka 6.1: Seznam testovacích souborů.

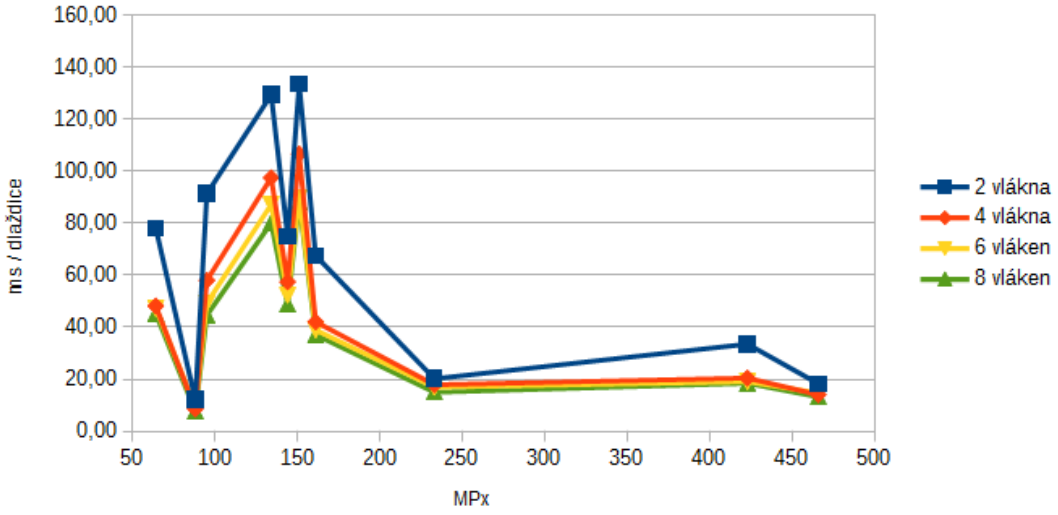
6.1 Rychlost běhu aplikace v závislostí na hardwaru

V kapitole 5 jsem z prvních testů vyhodnotil závislost rychlostí zpracování vzhledem k různým typům hardwaru. Obrázek 6.1 vyjadřuje vztah mezi průměrnou rychlostí načítání jedné dlaždice pro různé rozlišení s využitím jiného počtu jader CPU. Frekvence jader procesoru v tomto případě byla stejná. Test jsem vyhodnocoval ve virtuálním počítači s OS

[1] Katalog NASA http://visibleearth.nasa.gov/view_cat.php?categoryID=1484

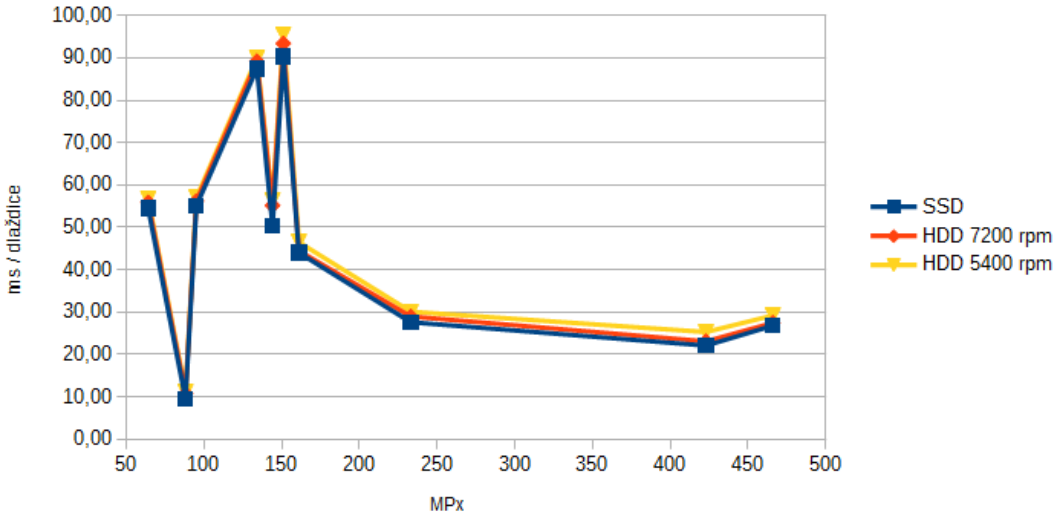
[2] Rastrová data z portálu Natural Earth http://www.naturalearthdata.com/downloads/10m-raster-data/

[3] Webová stránka firmy Klokan Technologies GmbH http://www.klokantech.com/



Obrázek 6.1: Závislost času zpracování dlaždice na rozlišení, testování počtu vláken. Formát vstupních rastrů je PNG.

Linux, který běžel na SSD disku. Omezování počtu jader procesoru by mělo charakterizovat různé počítače, které jsou v dnešní době dostupné.

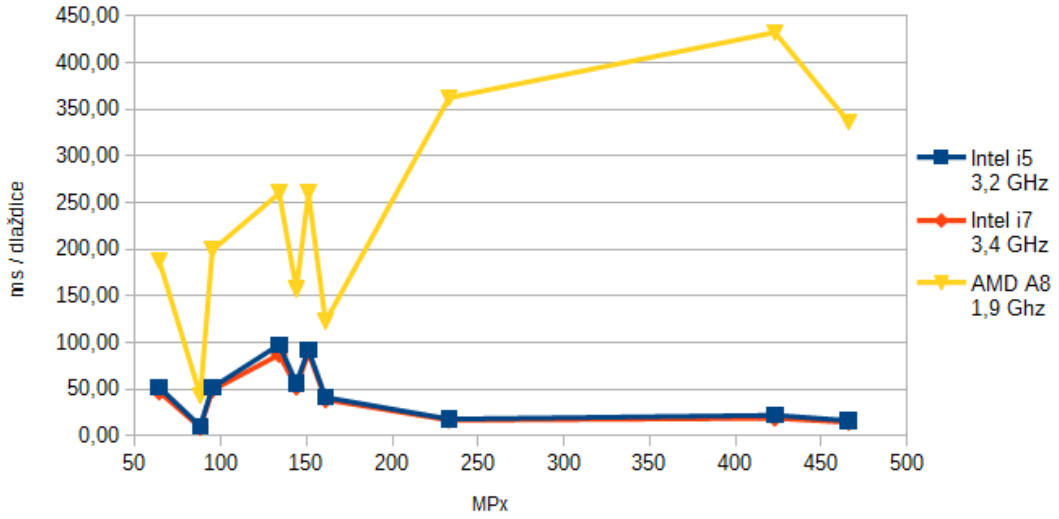


Obrázek 6.2: Závislost času zpracování dlaždice na rozlišení, testování různých disků. Formát vstupních rastrů je PNG.

Graf 6.2 zobrazuje rychlost zpracování aplikace, kde data byla uložená na jiném typu disku. Test jsem vyhodnocoval pod operačním systémem Windows 7. Procesor testovacího počítače byl Intel i7-3770 s frekvencí 3.40 GHz a 8 samostatně pracujících vláken.

Závislost rychlosti aplikace na frekvencí procesorů je znázorněná v grafu 6.3. Test jsem vyhodnocoval pod operačním systémem Windows 7. Procesory testovacích počítačů byly Intel i7-3770 s frekvencí 3,40 GHz, Intel i5-3470 s frekvencí 3,20 GHz a AMD A8-4500M s frekvencí 1,9 GHz. Všechny testovací počítače používaly stejný disk s testovacími daty a 4 pracující vlákna.

Grafy pro formáty TIFF a JPEG mají podobný průběh. Všechny grafy naznačují, že



Obrázek 6.3: Závislost času zpracování dlaždice na rozlišení, testování různých procesorů. Formát vstupních rastrů je PNG.

existují ještě jiné kritéria, které ovlivňují rychlost zpracování. Paleta barev výrazně zjednoduší dekompresí obrázků a zároveň zmenší velikost souboru. Vstupní testovací soubor s rozlišením 12 000 x 12 000 (144 MPx) má pro velkou část obrazu jednu barvu - černou. Zpracování obrazu s jedním kanálem (stupně šedi, viz 161 MPx) je taktéž rychlejší.

6.2 Porovnání aplikace QtGDAL s existujícím řešením

Prohlížení rozměrných rastrů je pomocí projektů z kapitoly 3 obtížné, zejména pro formáty typu PNG nebo JPEG. Takové formáty musíme konvertovat na jiné formáty (např. pro využití projektu IIPImage bez knihovny GDAL). Druhým způsobem zobrazení rozměrných rastrů je převod souboru na dlaždice.

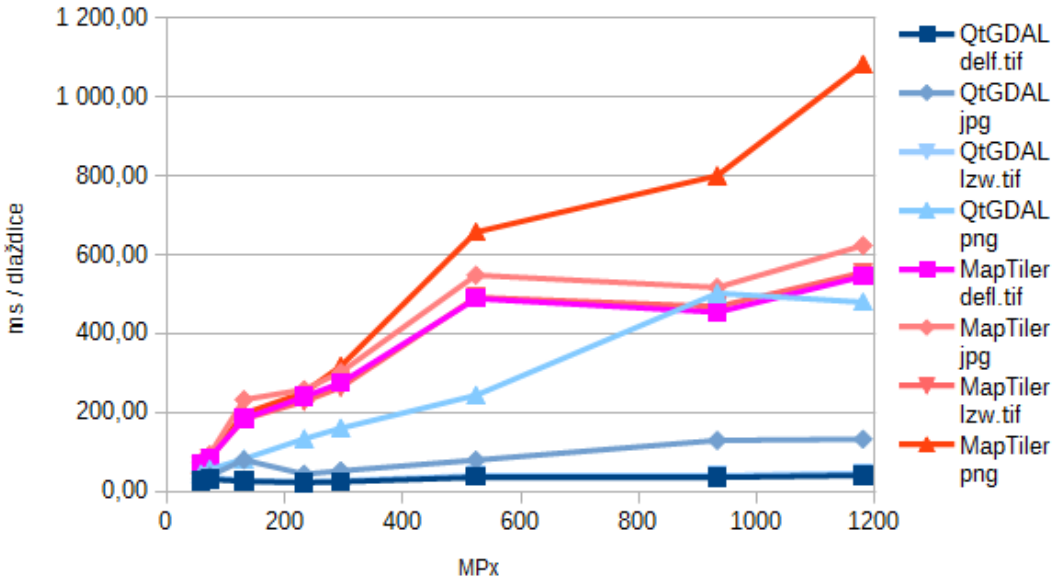
Jedním z nástrojů knihovny GDAL je `gdal2tiles.py` - skript v jazyce Python, který vygeneruje složku a malé dlaždice podle standardu TMS pro vstupní soubor. Nástroj poskytuje tři profily pro vytváření dlaždic: *mercator* (kompatibilní s Google Maps), *geodetic* (kompatibilní s Google Earth) a *raster*. Profil *raster* produkuje stejný výsledek jako aplikace QtGDAL - dlaždice vstupního rastru bez geografické informace. Utilita `gdal2tiles.py` není optimalizovaná pro vícejádrové procesory. Autoři[4] této utility vytvořili komerční aplikaci *MapTiler*[5], která je napsaná v jazyce C a optimalizovaná pro rychlé zpracování rozměrných rastrů.

Rozhodl jsem se otestovat rychlost aplikace MapTiler na testovacích souborech z kapitoly 5. Využil jsem možnost aplikace převést vstupní soubor na dlaždice pomocí profilu *raster* a nastavil jsem výstup na formát MBTiles[6]. K testování jsem využil verzi MapTiler Pro, která má program pro prací z příkazového řádku. Testoval jsem rychlost běhu programu MapTiler a sledoval jsem velikost vytvářených souborů formátu MBTiles.

[4] Autorem nástroje je firma Klokan Technologies GmbH.

[5] Webová stránka aplikace MapTiler http://maptiler.com/.

[6] Formát MBTiles uchovává dlaždice v jedném souboru využitím SQLite databáze. Více informací na stránkách specifikace formátu https://www.mapbox.com/guides/an-open-platform/#mbtiles.

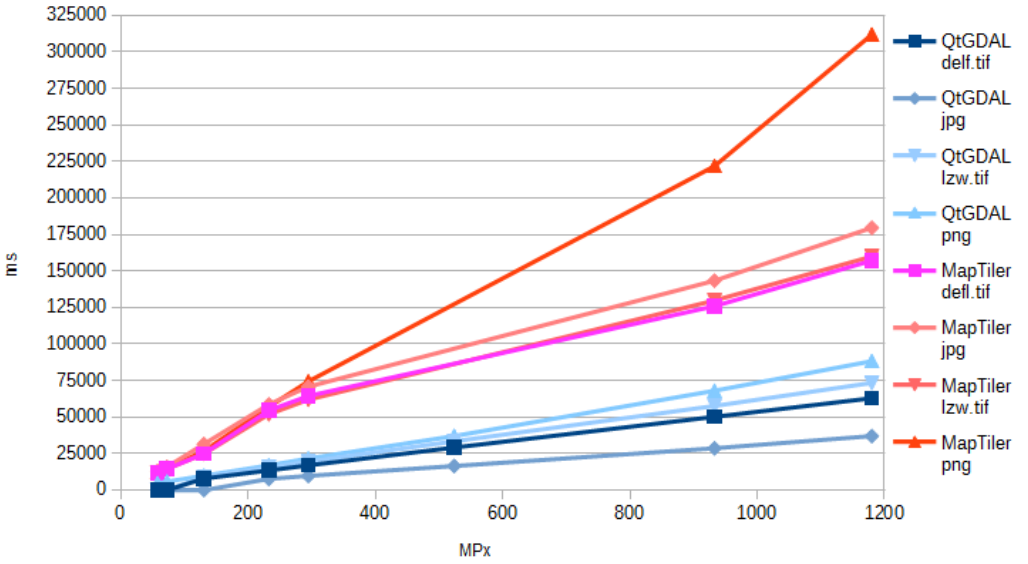


Obrázek 6.4: Porovnání časů zpracování jedné dlaždice pomocí aplikací MapTiler a QtGDAL. Počet dlaždic je stejný pro výpočet z celkového času běhu aplikací.

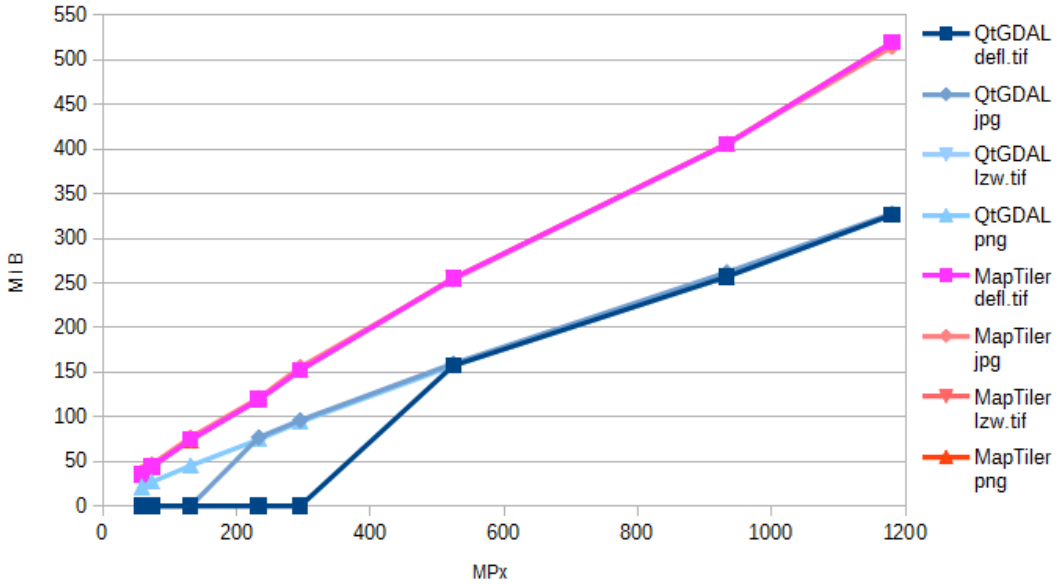
Graf 6.4 reprezentuje průměrný čas dlaždice při zpracování aplikaci QtGDAL a programem MapTiler s profilem *raster* pro různé rozlišení stejného testovacího obrazu[7]. Jelikož program MapTiler zpracuje celý soubor (na rozdíl od aplikace QtGDAL která zpracovává jenom částí obrazu na vyžádání), tak hodnoty grafu 6.4 představují čas na zpracování jedné dlaždice, když počet dlaždic je stejný jako při zpracování aplikací QtGDAL. Graf vyjadřuje poměr časů, když se zpracuje jenom určitý počet dlaždic. Reálný počet zpracovaných dlaždic programem MapTiler je vyšší (protože zpracuje celý soubor), v takovém případě je průměrný čas zpracování jedné dlaždice řádově menší než v případě aplikace QtGDAL (viz graf C.5 v příloze C.2 na straně 62). Poměr časů zpracování testovacích souborů obou aplikací vyjadřuje graf 6.5. Čas zpracování souboru aplikací QtGDAL je počet milisekund dynamického generování náhledů. Tyto grafy zobrazují skutečnost, že zobrazení rozměrných rastrů pomocí aplikace QtGDAL je rychlejší než generování všech dlaždic pro vstupní soubor programem MapTiler. Výhodou programu MapTiler je vytvoření souborů formátu MBTiles, který lze i pomocí aplikace QtGDAL zobrazovat rychleji než dynamické generování náhledů. Nevýhodou je nutnost předzpracování, které pro rozlišení nad 500 MPx převyšuje 100 sekund, tehdy doba čekání je více než 1,5 minuty. V případě rychlého zobrazení rozměrného souboru je výhodné použit aplikaci QtGDAL.

Porovnání velikostí vytvářených souborů obou testovaných aplikací je v grafu 6.6. Aplikace MapTiler vytváří stejně velké soubory pro libovolný vstupní formát stejného rozlišení (proto jsou v grafu překryté všechny 4 vstupní formáty). Aplikace QtGDAL s optimalizací rychlostí vytváří náhledy jenom pokud to je potřeba – viz v grafu formát TIFF s kompresí Deflate, kde soubor s rozlišením do 300 MPx nemá žádný pomocný soubor. Graf 6.7 reprezentuje poměr velikostí vytvořených souborů pro vstupní rastr s rozlišením 1 180 MPx. Aplikace MapTiler vytváří větší soubory MBTiles, nicméně pomocné soubory náhledů generované aplikací QtGDAL nejsou použitelné bez vstupního souboru.

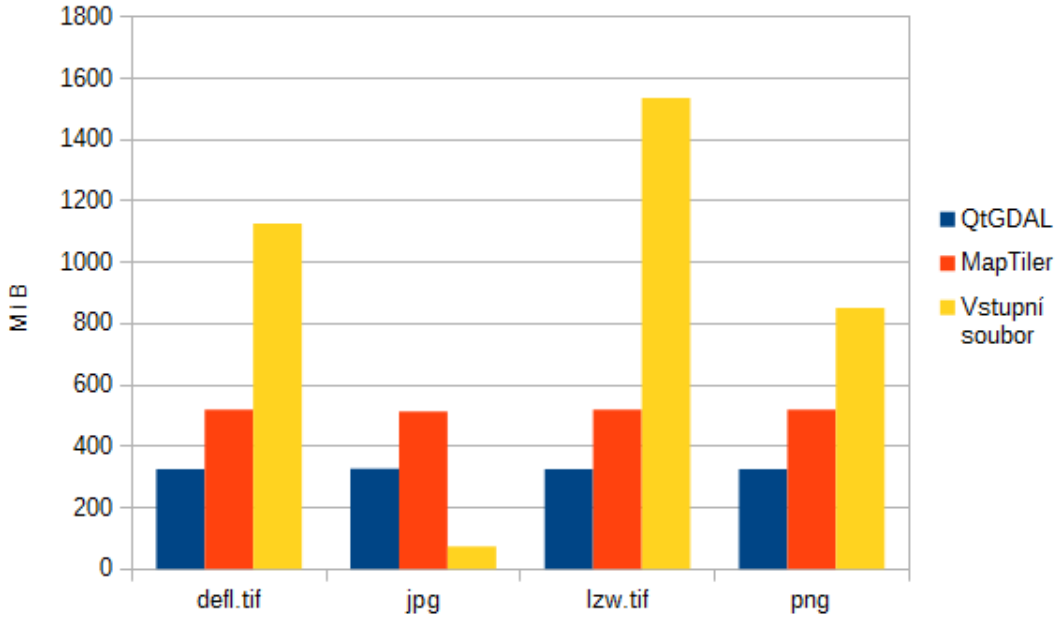
[7]Testovací obraz byl zmenšován na menší rozlišení, čímž se snaží poskytnout stejná pixelová data.



Obrázek 6.5: Porovnání časů zpracování testovacích souborů pomocí aplikací MapTiler a QtGDAL.



Obrázek 6.6: Porovnání velikostí generovaných souborů při zpracovávání rastrů pomocí aplikací MapTiler a QtGDAL.



Obrázek 6.7: Porovnání velikostí vytvořených souborů při zpracovávání vstupu s rozlišením 1 180 MPx pomocí aplikací MapTiler a QtGDAL.

Kapitola 7

Závěr

Tato diplomová práce se zaobírá zobrazováním rozměrných rastrových geodat. Cílem bylo vytvořit aplikaci, která bude optimalizována pro rychlé zpracování obrazu.

V první kapitole jsem popsal protokoly používané pro zpřístupnění rozměrných rastrových dat. Zaměřil jsem se na starší standardizované protokoly, IIP, Web Map Service a Web Map Tile Service a seznámil jsem se i s novým protokolem IIIF. Následuje kapitola, která popisuje projekty s otevřeným kódem a jejích základní principy přístupu k rozměrným rastrovým datům. Kapitola 4 popisuje návrh a implementaci aplikace, která pomocí knihovny GDAL zobrazuje rozměrná rastrová data. V podkapitole 4.4 jsem popsal implementaci rozšíření podpory GeoPDF pomocí open-source renderovacího jádra PDFium. V další kapitole jsem popsal způsob testování a optimalizaci na základě výsledků testů pro zrychlení zpracování obrazových dat. Poslední kapitola vyhodnocuje aplikaci a porovnává vlastnosti s již existujícími řešeními.

Experimentální testování aplikace ukázalo, že aplikace není optimalizovaná vzhledem k modelu barev vstupních souborů. Vhodným rozšířením optimalizace může být taktéž závislost na histogramu vstupního rastru.

Rozšíření podpory GeoPDF pomocí open-source knihovny je veřejně dostupné[1] a bude snaha o vložení podpory do jádra knihovny GDAL. Implementace podpory GeoPDF je aktuálně využitá firmou Klokan Technologies v zpřístupněné knihovně GDAL v systému Docker[2].

[1] Větev `pdfium-master` klonu repozitáře knihovny GDAL https://github.com/klokantech/gdal/tree/pdfium-master

[2] GDAL in Docker http://blog.klokantech.com/2015/02/gdal-in-docker-install-run-with-single.html

Literatura

[1] Albert, L., Kulp, D., Mutz, A. aj. *FlashPix Format Specification, Version 1.0.* [b.m.]: Eastman Kodak Company, 1996. 140 p.

[2] Albert, L., Kulp, D., Mutz, A. aj. *Internet Imaging Protocol, Version 1.0.5.* [b.m.]: Hewlett Packard Company, Live Picture, Inc., Eastman Kodak Company, 1997. 75 p.

[3] Beaujardiere, J. de la (ed.). *OpenGIS Web Map Service Implementation Specification, Version 1.3.0.* [b.m.]: Open Geospatial Consortium Inc., 2006. 85 p.

[4] Berners Lee, T. *RFC 2396: Uniform Resource Identifiers (URI).* [b.m.]: MIT, 1998. Dostupné z: <http://www.rfc-archive.org/getrfc.php?rfc=2396>.

[5] Broder, A. *About IIIF - IIIF — International Image Interoperability Framework* [online]. 15. 5. 2014 [cit. 2015-01-12]. Dostupné z: <http://iiif.io/about.html>.

[6] Cid, A. A. a Noonburg, D. *Poppler - hlavní stránka projektu.* 7. 3. 2015 [cit. 2015-05-05]. Dostupné z: <http://poppler.freedesktop.org/>.

[7] Cupitt, J. *History - VipsWiki* [online]. 3. 2. 2012 [cit. 2015-01-12]. Dostupné z: <http://www.vips.ecs.soton.ac.uk/index.php?title=History>.

[8] Garthwaite, E. Google throws PDFium into the open source community. *ITProPortal* [online]. 19. 6. 2014 [cit. 2015-05-05]. Dostupné z: <http://www.itproportal.com/2014/06/19/google-throws-pdfium-into-the-open-source-community/>.

[9] Kolektiv autorů. *V8 JavaScript Engine - hlavní stránka projektu* [online]. [cit. 2015-05-05]. Dostupné z: <https://code.google.com/p/v8/>.

[10] Martinez, K., Perry, S. T. a Cupitt, J. Object browsing using the Internet Imaging Protocol. *Computer Networks.* 2000, vol. 33, 1-6. P. 803–810.

[11] Masó, J., Pomakis, K. a Julia, N. (ed.). *OpenGIS Web Map Tile Service Implementation Standard, Version 1.0.0.* [b.m.]: Open Geospatial Consortium Inc., 2010. 129 p.

[12] Pillay, R. *IIPImage - History.* [cit. 2015-01-12]. Dostupné z: <http://iipimage.sourceforge.net/documentation/history/>.

[13] Schepis, A. *Aschepis/pdfium* [online]. 9. 9. 2014 [cit. 2015-05-05]. Dostupné z: <https://github.com/aschepis/pdfium>.

[14] Seichter, D. *PoDoFo - hlavní stránka projektu.* [cit. 2015-05-05]. Dostupné z: <http://podofo.sourceforge.net/about.html>.

[15] Simone, S. D. Google Chrome PDF Engine is now Open Source. *InfoQ.com* [online]. 20. 6. 2014 [cit. 2015-05-05]. Dostupné z: <http://www.infoq.com/news/2014/06/google-chrome-pdf-engine-free>.

[16] Software, F., Google a developers chromium. *PDFium - hlavní stránka projektu.* květen 2014 [cit. 2015-05-05]. Dostupné z: <https://code.google.com/p/pdfium/>.

[17] Stroop, J. *Image API 2.0 - IIIF — International Image Interoperability Framework* [online]. 13. 6. 2014 [cit. 2015-01-12]. Dostupné z: <http://iiif.io/api/image/2.0/#order-of-implementation>.

Příloha A

IIIF - Dotaz na popis obrazu

A.1 Seznam názvu podporovaných funkcí

Sekce 2.4.1 **Parametry požadavku na obraz** popisuje formy a hodnoty parametrů akceptované protokolem IIIF ve verzi 2.0[1][17]. Podmnožina seznamu níže uvedených řetězců se může nastavit pro hodnotu vlastností `/profile/supports`.

- `baseUriRedirect` - použití základní URI (bez parametrů požadavků) vrátí JSON odpověď pro dotaz na popis obrazu
- `canonicalLinkHeader` - obrazová odpověď obsahuje HTTP hlavičku s kanonickým URI
- `cors` - Cross-Origin Resource Sharing hlavička je v každé odpovědi serveru
- `jsonldMediaType` - server podporuje JSON-LD formát
- `mirroring` - zrcadlení podle vertikální ose
- `profileLinkHeader` - obrazová odpověď obsahuje HTTP hlavičku s odkazem na definici profilu
- `regionByPct` - možnost specifikovat požadovanou oblast v procentech
- `regionByPx` - možnost specifikovat požadovanou oblast v pixelech
- `rotationArbitrary` - rotace všech uhlů (nejen násobky 90) v rozmezí 0 - 360 stupňů
- `rotationBy90s` - rotace jen pomocí násobků 90 stupňů
- `sizeAboveFull` - výstupní velikost může být větší než originální obraz
- `sizeByWhListed` - je možné dotazovat se na přesně specifikované velikosti v pixelech, uvedené v JSON odpovědi pole `sizes`
- `sizeByForcedWh` - výstupní velikost zapsaná pomocí `!w,h`
- `sizeByH` - výstupní velikost zapsaná pomocí `,h`
- `sizeByPct` - výstupní velikost zapsaná pomocí `pct:n`

[1] http://iiif.io/api/image/2.0/#image-request-parameters

- `sizeByW` - výstupní velikost zapsaná pomocí `w,`
- `sizeByWh` - výstupní velikost zapsaná pomocí `w,h`

A.2 Příklad JSON odpovědi k dotazu na popis obrazu

Příklad k formálně správné odpovědi na dotaz
`http://www.example.org/image-service/abcd1234/1E34750D-38DB/info.json`

```
{
  "@context" : "http://iiif.io/api/image/2/context.json",
  "@id" : "http://www.example.org/image-service/abcd1234/1E34750D-38DB",
  "protocol" : "http://iiif.io/api/image",
  "width" : 6000,
  "height" : 4000,
  "sizes" : [
    {"width" : 150, "height" : 100},
    {"width" : 600, "height" : 400},
    {"width" : 3000, "height": 2000}
  ],
  "tiles": [
    {"width" : 512, "scaleFactors" : [1,2,4,8,16]}
  ],
  "profile" : [
    "http://iiif.io/api/image/2/level2.json",
    {
      "formats" : [ "gif", "pdf" ],
      "qualities" : [ "color", "gray" ],
      "supports" : [
          "canonicalLinkHeader",
          "rotationArbitrary",
          "profileLinkHeader",
          "http://example.com/feature/"
      ]
    }
  ],
  "service" : {
    "@context": "http://iiif.io/api/annex/service/physdim/1/context.json",
    "profile": "http://iiif.io/api/annex/service/physdim",
    "physicalScale": 0.0025,
    "physicalUnits": "in"
  }
}
```

Příloha B

PDFium - rozdílové soubory pro podporu v GDAL

B.1 Zdrojové a konfigurační soubory knihovny

```
diff --git a/fpdfsdk/src/fsdk_mgr.cpp b/fpdfsdk/src/fsdk_mgr.cpp
--- a/fpdfsdk/src/fsdk_mgr.cpp
+++ b/fpdfsdk/src/fsdk_mgr.cpp
@@ -214,8 +214,12 @@ FX_SYSTEMTIME CFX_SystemHandler::GetLoca

 CJS_RuntimeFactory* GetJSRuntimeFactory()
 {
+#ifdef _V8_SUPPORT_
   static CJS_RuntimeFactory s_JSRuntimeFactory;
   return &s_JSRuntimeFactory;
+#else
+    return NULL;
+#endif
 }

 CPDFDoc_Environment::CPDFDoc_Environment(CPDF_Document* pDoc) :
@@ -233,8 +237,10 @@ CPDFDoc_Environment::CPDFDoc_Environment


   m_pJSRuntimeFactory = NULL;
+#ifdef _V8_SUPPORT_
   m_pJSRuntimeFactory = GetJSRuntimeFactory();
   m_pJSRuntimeFactory->AddRef();
+#endif
 }

 CPDFDoc_Environment::~CPDFDoc_Environment()
@@ -245,9 +251,11 @@ CPDFDoc_Environment::~CPDFDoc_Environmen
     delete m_pIFormFiller;
     m_pIFormFiller = NULL;
   }
+#ifdef _V8_SUPPORT_
   if(m_pJSRuntime && m_pJSRuntimeFactory)
     m_pJSRuntimeFactory->DeleteJSRuntime(m_pJSRuntime);
   m_pJSRuntimeFactory->Release();
+#endif

   if(m_pSysHandler)
   {
```

Zdrojový kód B.1: Změny v souboru `fpdfsdk/src/fsdk_mgr.cpp`, 1. část

```
@@ -272,12 +280,16 @@ CPDFDoc_Environment::~CPDFDoc_Environmen

 IFXJS_Runtime* CPDFDoc_Environment::GetJSRuntime()
 {
+#ifdef _V8_SUPPORT_
   if(!IsJSInitiated())
     return NULL;
   assert(m_pJSRuntimeFactory);
   if(!m_pJSRuntime)
     m_pJSRuntime = m_pJSRuntimeFactory->NewJSRuntime(this);
   return m_pJSRuntime;
+#else
+    return NULL;
+#endif
 }

 CPDFSDK_AnnotHandlerMgr* CPDFDoc_Environment::GetAnnotHandlerMgr()
```

Zdrojový kód B.2: Změny v souboru `fpdfsdk/src/fsdk_mgr.cpp`, 2. část

```
diff --git a/javascript.gypi b/javascript.gypi
new file mode 100644
index 0000000..972521f
--- /dev/null
+++ b/javascript.gypi
@@ -0,0 +1,82 @@
+{
+ 'targets': [
+  {
+   'target_name': 'javascript',
+   'type': 'static_library',
+   'include_dirs': [
+    '<(DEPTH)/v8',
+    '<(DEPTH)/v8/include',
+   ],
+   'dependencies': [
+    '<(DEPTH)/v8/tools/gyp/v8.gyp:v8',
+   ],
+   'export_dependent_settings': [
+    '<(DEPTH)/v8/tools/gyp/v8.gyp:v8',
+   ],
+   'ldflags': [ '-L<(PRODUCT_DIR)',],
+   'sources': [
+    'fpdfsdk/include/javascript/app.h',
+    'fpdfsdk/include/javascript/color.h',
+    'fpdfsdk/include/javascript/console.h',
+    'fpdfsdk/include/javascript/Consts.h',
+    'fpdfsdk/include/javascript/Document.h',
+    'fpdfsdk/include/javascript/event.h',
+    'fpdfsdk/include/javascript/Field.h',
+    'fpdfsdk/include/javascript/global.h',
+    'fpdfsdk/include/javascript/Icon.h',
+    'fpdfsdk/include/javascript/IJavaScript.h',
+    'fpdfsdk/include/javascript/JavaScript.h',
+    'fpdfsdk/include/javascript/JS_Console.h',
+    'fpdfsdk/include/javascript/JS_Context.h',
+    'fpdfsdk/include/javascript/JS_Define.h',
+    'fpdfsdk/include/javascript/JS_EventHandler.h',
+    'fpdfsdk/include/javascript/JS_GlobalData.h',
+    'fpdfsdk/include/javascript/JS_Module.h',
+    'fpdfsdk/include/javascript/JS_Object.h',
+    'fpdfsdk/include/javascript/JS_Runtime.h',
+    'fpdfsdk/include/javascript/JS_Value.h',
+    'fpdfsdk/include/javascript/PublicMethods.h',
+    'fpdfsdk/include/javascript/report.h',
+    'fpdfsdk/include/javascript/resource.h',
+    'fpdfsdk/include/javascript/util.h',
+    'fpdfsdk/src/javascript/app.cpp',
+    'fpdfsdk/src/javascript/color.cpp',
+    'fpdfsdk/src/javascript/console.cpp',
+    'fpdfsdk/src/javascript/Consts.cpp',
+    'fpdfsdk/src/javascript/Document.cpp',
+    'fpdfsdk/src/javascript/event.cpp',
+    'fpdfsdk/src/javascript/Field.cpp',
+    'fpdfsdk/src/javascript/global.cpp',
+    'fpdfsdk/src/javascript/Icon.cpp',
+    'fpdfsdk/src/javascript/JS_Context.cpp',
+    'fpdfsdk/src/javascript/JS_EventHandler.cpp',
+    'fpdfsdk/src/javascript/JS_GlobalData.cpp',
+    'fpdfsdk/src/javascript/JS_Object.cpp',
+    'fpdfsdk/src/javascript/JS_Runtime.cpp',
+    'fpdfsdk/src/javascript/JS_Value.cpp',
+    'fpdfsdk/src/javascript/PublicMethods.cpp',
+    'fpdfsdk/src/javascript/report.cpp',
+    'fpdfsdk/src/javascript/util.cpp',
+   ],
+  },
+  {
+   'target_name': 'jsapi',
+   'type': 'static_library',
+   'dependencies': [
+    '<(DEPTH)/v8/tools/gyp/v8.gyp:v8',
+   ],
+   'export_dependent_settings': [
+    '<(DEPTH)/v8/tools/gyp/v8.gyp:v8',
+   ],
+   'include_dirs': [
+    '<(DEPTH)/v8',
+    '<(DEPTH)/v8/include',
+   ],
+   'ldflags': [ '-L<(PRODUCT_DIR)',],
+   'sources': [
+    'fpdfsdk/include/jsapi/fxjs_v8.h',
+    'fpdfsdk/src/jsapi/fxjs_v8.cpp',
+   ],
+  }
+ ]
+}
```

Zdrojový kód B.3: Nový soubor `javascript.gypi`

```
--- a/samples/samples.gyp
+++ b/samples/samples.gyp
@@ -3,16 +3,27 @@
 # found in the LICENSE file.

 {
+  'variables': {
+    'pdf_use_v8%': 0,
+  },
   'target_defaults': {
     'type': 'executable',
     'dependencies': [
       '../pdfium.gyp:pdfium',
-      '<(DEPTH)/v8/tools/gyp/v8.gyp:v8_libplatform',
+    ],
+    'conditions': [
+      ['pdf_use_v8==1', {
+        'dependencies': [
+          '<(DEPTH)/v8/tools/gyp/v8.gyp:v8_libplatform',
+        ],
+        'include_dirs': [
+          '<(DEPTH)/v8',
+          '<(DEPTH)/v8/include',
+        ],
+      }]
     ],
     'include_dirs': [
       '<(DEPTH)',
-      '<(DEPTH)/v8',
-      '<(DEPTH)/v8/include',
     ],
   },
   'targets': [
```

Zdrojový kód B.4: Změny v souboru `samples/samples.gyp`

```
diff --git a/pdfium.gyp b/pdfium.gyp
index 15f942d..98496a7 100644
--- a/pdfium.gyp
+++ b/pdfium.gyp
@@ -1,12 +1,20 @@
 {
   'variables': {
     'pdf_use_skia%': 0,
+    'pdf_use_v8%': 0,
     'conditions': [
       ['OS=="linux"', {
         'bundle_freetype%': 0,
       }],
     ],
   },
+  'conditions': [
+    ['pdf_use_v8==1', {
+      'includes': [
+        'javascript.gypi'
+      ]
+    }]
+  ],
   'target_defaults': {
     'defines' : [
       '_FPDFSDK_LIB',
@@ -23,6 +31,12 @@
       ['pdf_use_skia==1', {
         'defines': ['_SKIA_SUPPORT_'],
       }],
+      ['pdf_use_v8==1', {
+        'defines': ['_V8_SUPPORT_'],
+        'includes': [
+          'javascript.gypi'
+        ]
+      }],
       ['OS=="linux"', {
         'conditions': [
           ['target_arch=="x64"', {
@@ -55,10 +69,16 @@
         'fxcrt',
         'fxedit',
         'fxge',
-        'javascript',
-        'jsapi',
         'pdfwindow',
       ],
+      'conditions': [
+        ['pdf_use_v8==1', {
+          'dependencies': [
+            'javascript',
+            'jsapi'
+          ]
+        }]
+      ],
       'ldflags': [ '-L<(PRODUCT_DIR)',],
       'sources': [
         'fpdfsdk/include/fpdfdoc.h',
@@ -717,84 +737,6 @@
       ],
     },
     {
-      'target_name': 'javascript',
...
-    },
-    {
-      'target_name': 'jsapi',
...
-    },
-    {
       'target_name': 'formfiller',
       'type': 'static_library',
       'ldflags': [ '-L<(PRODUCT_DIR)',],
```

Zdrojový kód B.5: Změny v souboru `pdfium.gyp`

B.2 Úprava souboru pro kompilací v Operačním Systému Windows

```
diff --git a/build/standalone.gypi b/build/standalone.gypi
index 645b9f2..f28cb8a 100644
--- a/build/standalone.gypi
+++ b/build/standalone.gypi
@@ -161,7 +161,7 @@
     ],
     'msvs_cygwin_dirs': ['<(DEPTH)/v8/third_party/cygwin'],
     'msvs_configuration_attributes': {
-      'OutputDirectory': '<(DEPTH)\\build\\$(ConfigurationName)',
+      'OutputDirectory': '<(DEPTH)\\build\\$(ConfigurationName)-$(PlatformName)',
       'IntermediateDirectory': '$(OutDir)\\obj\\$(ProjectName)',
       'CharacterSet': '1',
     },
```

Zdrojový kód B.6: Změny v souboru `build/standalone.gypi`

```
diff --git a/core/include/fpdfapi/fpdf_parser.h b/core/include/fpdfapi/fpdf_parser.h
index 4b91802..ce4b3a5 100644
--- a/core/include/fpdfapi/fpdf_parser.h
+++ b/core/include/fpdfapi/fpdf_parser.h
@@ -411,7 +411,7 @@ class CPDF_Parser FX_FINAL : public IPDF_DocParser
 public:

     CPDF_Parser();
-    ~CPDF_Parser() override;
+    ~CPDF_Parser() FX_OVERRIDE;

     FX_DWORD     StartParse(FX_LPCSTR filename, FX_BOOL bReParse = FALSE);

diff --git a/core/include/fxcrt/fx_basic.h b/core/include/fxcrt/fx_basic.h
index a3cf585..2c61fb1 100644
--- a/core/include/fxcrt/fx_basic.h
+++ b/core/include/fxcrt/fx_basic.h
@@ -271,7 +271,7 @@ class CFX_FileBufferArchive : public IFX_BufferArchive, public CFX_Object
 {
 public:
     CFX_FileBufferArchive(FX_STRSIZE size = 32768);
-    ~CFX_FileBufferArchive() override;
+    ~CFX_FileBufferArchive() FX_OVERRIDE;
     virtual void     Clear();

     FX_BOOL          AttachFile(IFX_StreamWrite *pFile, FX_BOOL bTakeover = FALSE);
```

Zdrojový kód B.7: Změny v souborech `core/include/fpdfapi/fpdf_parser.h` a `core/include/fxcrt/fx_basic.h`

```
diff --git a/core/src/fxcodec/fx_libopenjpeg/libopenjpeg20/opj_config_private.h b/...
index 7d7e3ea..2367d50 100644
--- a/core/src/fxcodec/fx_libopenjpeg/libopenjpeg20/opj_config_private.h
+++ b/core/src/fxcodec/fx_libopenjpeg/libopenjpeg20/opj_config_private.h
@@ -5,7 +5,11 @@
 // Original code copyright 2014 Foxit Software Inc. http://www.foxitsoftware.com

 /* create opj_config_private.h for CMake */
-#define OPJ_HAVE_INTTYPES_H  1
+#ifndef _WIN32
+#ifndef OPJ_HAVE_INTTYPES_H
+#define OPJ_HAVE_INTTYPES_H
+#endif
+#endif

 #define OPJ_PACKAGE_VERSION "2.1.0"
```

Zdrojový kód B.8: Změny v souboru `core/src/fxcodec/fx_libopenjpeg/libopenjpeg20/opj_config_private.h`

```
diff --git a/fpdfsdk/include/fpdfview.h b/fpdfsdk/include/fpdfview.h
index f9df408..c2eaace 100644
--- a/fpdfsdk/include/fpdfview.h
+++ b/fpdfsdk/include/fpdfview.h
@@ -112,7 +112,12 @@ typedef const FS_RECTF*  FS_LPCRECTF;
 #define STDCALL __stdcall
 #else
 #define DLLEXPORT
+#if defined(_WIN32) && !defined(_WIN64)
+// On Windows Win32 (not x64) requires __stdcall convention
+#define STDCALL __stdcall
+#else
 #define STDCALL
+#endif  // !defined(_WIN64)
 #endif

 extern const char g_ExpireDate[];
```

Zdrojový kód B.9: Změny v souboru `fpdfsdk/include/fpdfview.h`

```
diff --git a/pdfium.gyp b/pdfium.gyp
index 98496a7..6943ef9 100644
--- a/pdfium.gyp
+++ b/pdfium.gyp
@@ -50,8 +50,13 @@
       }],
     ],
     'msvs_disabled_warnings': [
-      4005, 4018, 4146, 4333, 4345, 4267
+      4005, 4018, 4146, 4333, 4345, 4267, 4715, 4244
     ],
+    'msvs_settings': {
+      'VCCLCompilerTool': {
+        'WarnAsError': 'false',
+      },
+    },
   },
   'targets': [
     {
```

Zdrojový kód B.10: Změny v souboru `pdfium.gyp`

Příloha C

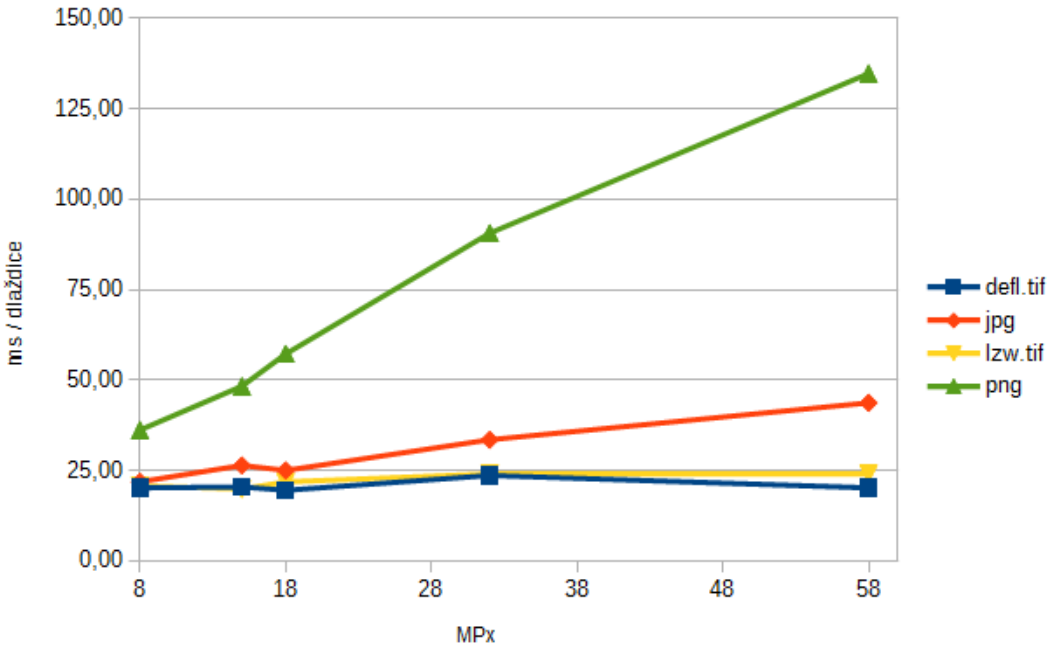
Dodatečné grafy pro optimalizaci a experimentální testování

Příloha obsahuje několik grafů, které jsou využité při optimalizaci aplikace QtGDAL (podkapitola C.1) nebo lépe prezentují výsledký experimentálního testování aplikace (podkapitola C.2).

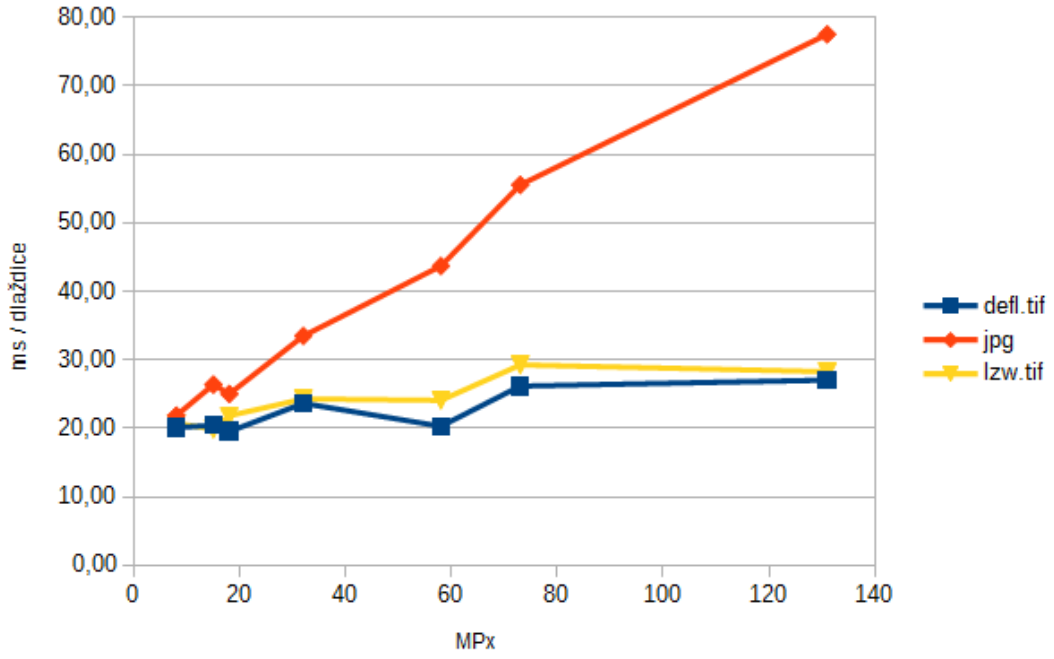
C.1 Optimalizace rychlosti zpracování aplikace QtGDAL

Kapitola 5 (na straně 34) obsahuje popis a využití těchto grafů k optimalizaci aplikace QtGDAL.

C.1.1 Načítání nativního obrázku

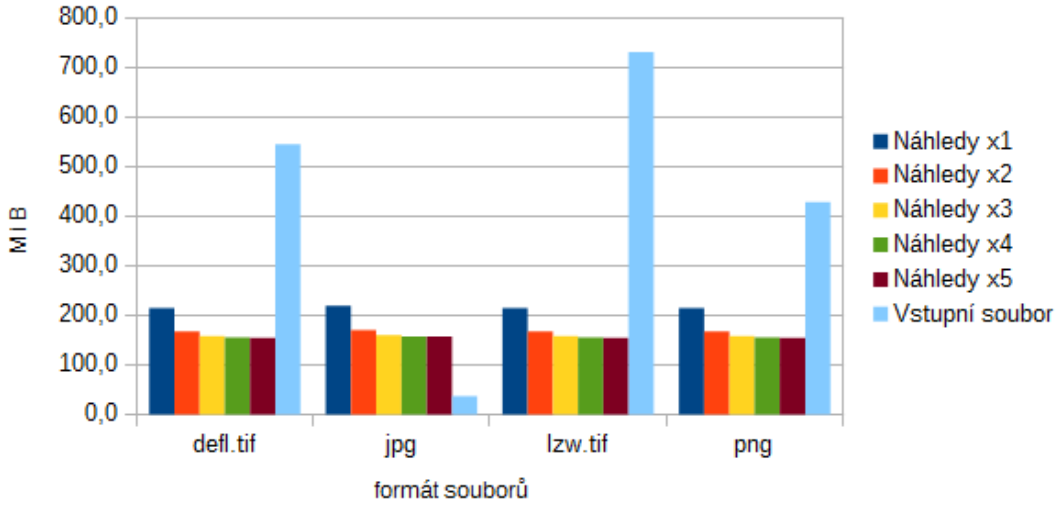


Obrázek C.1: Časová závislost načítání dlaždice na rozlišení do 60 MPx.

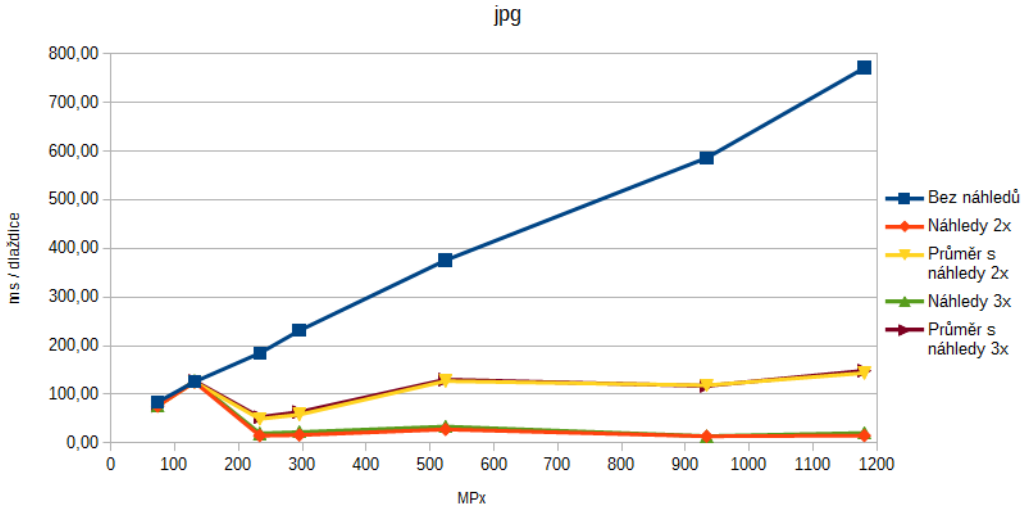


Obrázek C.2: Časová závislost načítání dlaždice na rozlišení do 200 MPx.

C.1.2 Generování náhledů



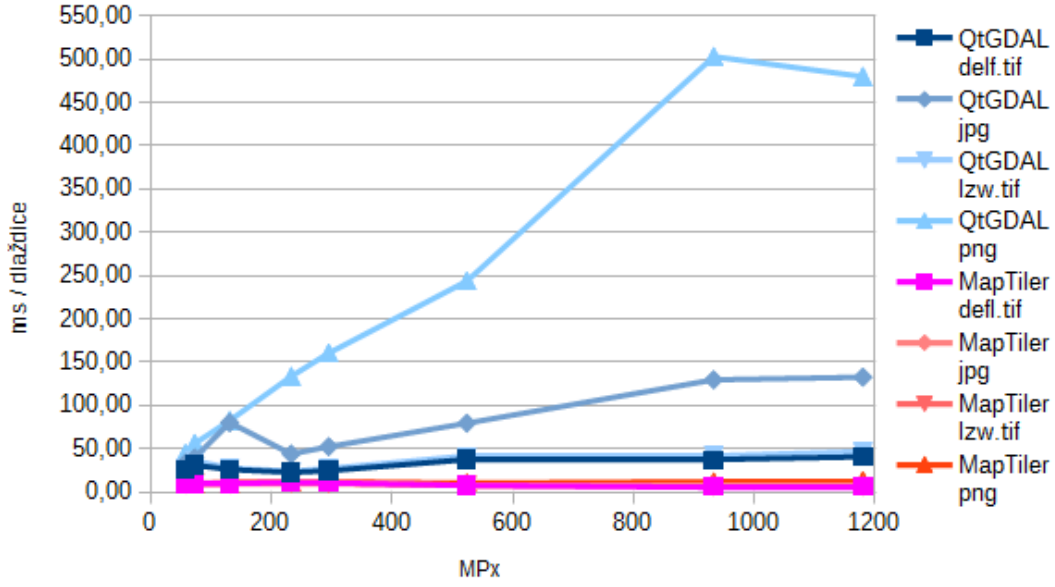
Obrázek C.3: Poměr velikosti pomocných souborů s vytvořenými náhledy a vstupního souboru s rozlišením 524 MPx.



Obrázek C.4: Časová závislost zpracování dlaždice obrazu (s využitím náhledů) na rozlišení vstupního obrazu (formátu JPG).

C.2 Experimentální testování aplikace QtGDAL

C.2.1 Porovnání aplikace QtGDAL s existujícími řešeními



Obrázek C.5: Porovnání časů zpracování jedné dlaždice pomocí aplikací MapTiler a QtGDAL. Počet dlaždic odpovídá reálnému zpracování souborů.