UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Obrazová analýza meteorologických radarových dat

Bc. Tomáš Dostalík

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: .....
Osobní číslo: .....
Studijní program: **N2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: .....
.....
Zadávající katedra: **Katedra informačních technologií**

Zásady pro vypracování:

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

MURPHY, M.,L. Android 2 - Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011. 371 s. EAN 9788025131947.
LEWIS, H. R., DENENBERG, L. Data structures and their algorithms. Berkley, Adison-Wesley, 1997.
Android developers Homepage [online]. 2011 [cit. 2011-10-07]. Dostupné z WWW: http://developer.android.com/.

Vedoucí diplomové práce: .....
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2011**
Termín odevzdání diplomové práce: **18. května 2012**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 27.5.2013 Bc. Tomáš Dostalík

Poděkování

Děkuji vedoucímu mé diplomové práce Ing. Petru Veselému za ochotu, cenné rady, připomínky a metodické vedení práce. Dále děkuji Ing. Vladimíru Harbichovi a Bc. Michalu Obešlovi ze společnosti T – CZ, a.s., za umožnění zpracování daného tématu v prostředí firmy.

Anotace

Práce pojednává o problematice získání a zpracování meteorologických informací z letištního přehledového radaru pro potřeby systémů řízení letového provozu a do jimi používaných formátů. V teoretické části je proveden rozbor problematiky radiolokace, různých typů radarů a získávání potřebných informací. Dále je navržen postup zpracování těchto informací, popsány algoritmy, pomocí kterých je možno dosáhnout požadovaného cíle. Praktická část se pak věnuje několika způsobům implementace popsaných algoritmů a dále posuzování jejich vhodnosti pro cíle této práce.

Klíčová slova

radiolokace, radar, meteorologie, filtrace obrazu, detekce objektů v obraze, zjednodušování linií

Title

Image analysis of meteorological radar data.

Annotation

This thesis deals with the problems of obtaining and processing meteorological information from airport surveillance radar for the use of air traffic management systems and formats they use. The theoretical part discusses the issue of radiolocation, various types of radars and obtaining the necessary information. Further, the aproach to processing the information is designed, algorithms by which it is possible to achieve the desired objectives are described. The practical part is devoted to several implementations of described algorithms and assess their suitability for the goals of this work.

Keywords

radiolocation, radar, meteorology, image filtering, contour tracing, simplifying polylines

Obsah

Seznam zkratek

ATC	Air traffic control
ATM	Air traffic management
SES	Single European Sky
IDE	Integrated Development Enviroment
SDK	Software Development Kit
ISO	International OrganizationforStandardisation
JVM	Java Virtual Machine

Seznam obrázků

Seznam tabulek

Úvod

Historie radiolokace má své počátky v první polovině 20. století, kdy také byly uskutečněny první úspěšné pokusy o měření vzdálenosti pomocí elektromagnetických vln. Vývoj radarů blízkých těm, které používáme dnes, je úzce svázán s velkým pokrokem v letectví ve 30. letech 20. století. Tedy dobou, ve které bylo vyrobeno první letadlo převážející cestující. Letecký průmysl od té doby prošel velkým vývojem a stal se z něj jeden z nejvýznamnějších způsobů dopravy vůbec. Největšího rozmachu se oblast osobní dopravy dočkala na přelomu 80. a 90. let. V souvislosti s dramatickým nárůstem uskutečněných letů a jejich zvyšujícím se zpožděním v Evropě vznikl projekt Single European Sky, tedy „Jednotné evropské nebe“. Vznikl především s cílem zvýšit kvalitu a bezpečnost letecké přepravy. V rámci tohoto projektu byly mimo jiné vytvořeny protokoly, definující formát přenášených dat mezi jednotlivými systémy, podílejícími se na řízení letového provozu. Jejich účelem je zjednodušit přenos důležitých informací mezi různorodými systémy. Jedna skupina těchto protokolů specifikuje formát předávání informací získávaných různými radarovými systémy, jejichž úkolem je v reálném čase prohledávat a identifikovat objekty ve vzdušném prostoru.

Mezi informacemi poskytovanými letištními přehledovými radary jsou kromě jiného také informace o aktuální meteorologické situaci nad sledovaným územím. Radary jsou tedy schopné detekovat kromě letadel a jiných pevných objektů také bouřkové mraky, déšť či mlhu. Cílem této práce je vytvoření postupu a nalezení vhodné kombinace algoritmů pro převod meteorologických informací poskytnutých přehledovým radarem do formátu specifikovaného v rámci projektu SES. V teoretické části je stručně rozebrána problematika radiolokace, získávání těchto dat, možnosti jejich reprezentace a postup jejich zpracování do cílové podoby, včetně vybraných algoritmů z oblasti počítačové grafiky a kartografie, využitelných k tomuto účelu.

Obsahem praktické části je implementace vybraných algoritmů na různých platformách, s cílem změření a vyhodnocení jejich vhodnosti ke zpracování meteorologických informací, především z pohledu jejich rychlosti.

1 Radiolokace

Radiolokací se nazývá proces rádiového vyhledávání objektů v prostoru a určování parametrů jejich pohybu, jako jsou např. rychlost či vzdálenost, a plochy za pomoci elektromagnetických vln. Zařízení k tomuto účelu určená se nazývají radiolokátory, neboli zkráceně radary. Slovo radar vzniklo zkrácením anglického výrazu „Radio Detection And Ranging“, tedy „Rádiová detekce předmětů a měření jejich vzdálenosti“. Mezi hledané objekty patří mimo jiné letadla, vozidla, rakety, geografické útvary, bouřkové mraky či jiné meteorologické útvary. [1] [2]

1.1 Historie

Nejdůležitějším způsobem využití radiolokačního principu je tzv. aktivní radiolokace, tedy ozařování cíle elektromagnetickou energií a zachytávání odraženého signálu a jeho následnou analýzou. Prvním, kdo veřejně publikoval nápad využít odrazu elektromagnetické energie ke zjišťování vzdálených předmětů, byl německý technik Christian Hülsmeyer. V roce 1904 v Kolíně nad Rýnem sestrojil, předvedl a patentoval svůj „Telemobiloskop“. Bohužel však toto zařízení neobsahovalo zesilovač, a proto bylo schopno detekovat pouze velké vodivé předměty, jako lodě nebo vlaky, na vzdálenost několika kilometrů. První měření vzdálenosti pomocí rádiových vln systémem s frekvenční modulací bylo uskutečněno roku 1924 ve Velké Británii v Oxfordu, fyziky Appletonem a Barnettem, při jejich pokusu o změření výšky ionosféry. Roku 1925 pokus opakovali Breit a Ture s využitím impulsové metody, dodnes používané v moderních radiolokačních systémech. [1]

Až do počátku třicátých let nebylo známo, že i malé cíle, jako např. osamocená letadla mohou být zdrojem měřitelných odrazů signálu. Údaje o této skutečnosti byly publikovány v Anglii a USA mezi lety 1931 a 1933. Nejrychleji postupoval vývoj právě v Anglii, která se cítila nejvíce ohrožena německým letectvem, a použití radiolokátorů pro jejich zjišťování se stalo prvořadou potřebou britské obrany. Již na jaře roku 1938 byla vybudována soustava výstražných radiolokátorů k obraně Londýna a ústí Temže. [2]

Obdobím nejrychlejšího rozvoje radiolokačních systému byla 2. světová válka. Nárůst rychlosti vojenských letadel, rozšiřování jejich akčního rádiusu a zdokonalení raketové techniky si vyžádalo konstrukci radiolokátorů s dosahem převyšujícím stovky kilometrů, budování rozsáhlých systému umožňujících spolehlivé zjištění a sledování pohybu letadel spolu s automatizovaným řízením systémů protivzdušné obrany. [2]

1.2 Radiolokace v dnešní době

Jedinou doménou radiolokačních systémů však vojenství dlouho nezůstalo, právě naopak. Velice brzy si našly své nezastupitelné místo i v mnoha oblastech civilního sektoru, zejména pak v letecké či lodní dopravě. Přehledové a sekundární lokátory sledují pohyb dopravních letadel na jejich trasách či v blízkosti letišť. Palubní radiolokátory slouží pro potřeby navigace a zjišťování vzdušných oblastí s nepříznivými povětrnostními

podmínkami a přesné přistávací systémy umožňují bezpečnou a spolehlivou navigaci letadel na přistávací dráhu i za špatných meteorologických podmínek a v noci. Mezi další oblasti využití radarů patří např. mapování neznámých oblastí, geologický či kosmický průzkum, sledování stavu a hustoty srážkové oblačnosti. V současné době se radary čím dál častěji instalují poblíž silnic k měření rychlosti projíždějících automobilů, či přímo do automobilů, kde slouží jako prvek pasivní bezpečnosti snižující riziko srážky s jinými automobily. [1]

V posledních letech se rozvíjejí především kombinované multipoziční a multisenzorové systémy, umožňující využití všech předností a potlačení nedostatků jednotlivých zařízení. [1]

1.3 Teorie radarů a radiolokace

Jak již bylo zmíněno výše, radary pracují s elektromagnetickým vlněním, které se šíří prostorem a které je vyzařováno a zachycováno jejich anténami. Používané frekvence signálu jsou radarům přidělovány na základě národních kmitočtových tabulek. Jejich volba také souvisí s akceptovatelnými rozměry antény, určením radaru a požadovaným dosahem. Kmitočet se pak pohybuje od desítek MHz až do několika set GHz. [1]

1.4 Typy radarů a jejich využití

Základní rozdělení radarů lze provést podle toho, zda samy vyzařují elektromagnetickou energii, či ne, a to do dvou hlavních skupin. Aktivní radary signál vyzařují, zatímco pasivní radary přijímají signál generovaný jinými objekty. Aktivní radary se dají dále rozdělit podle způsobu jejich činnosti na radary primární a sekundární. [1]

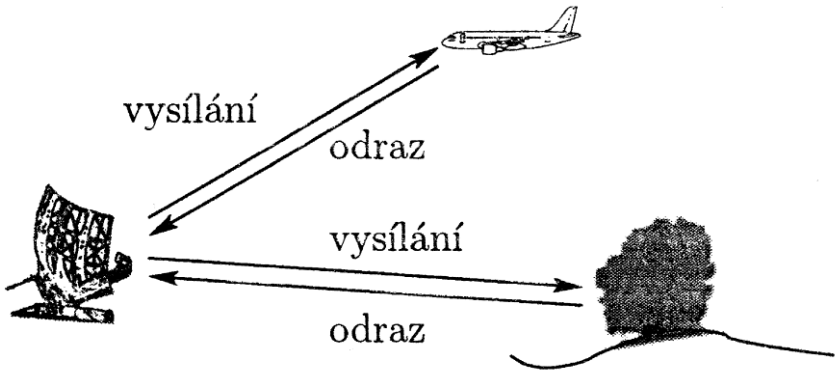
1.4.1 Primární radar

Principem činnosti primárního radaru je vyslání elektromagnetického signálu, který se šíří z antény do svého okolí. Odraženou energii tohoto signálu následně přijímá a vyhodnocuje. Analýzou odraženého signálu dokáže zjistit vzdálenost a rychlost cíle vztaženou k radaru. Princip fungování primárního radaru ilustruje obrázek 1. [1]

Vzdálenost cíle je určována ze zpoždění mezi vyslaným a zachyceným odraženým signálem. Úhlová poloha cíle je stanovena pomocí směrových vlastností antény. Vzájemná rychlost radaru a detekovaného předmětu je vypočtena z Dopplerova posunu frekvence zachyceného signálu oproti signálu vyzářenému, viz kapitola 1.5. [1]

K oblastem činností využívajícím primárních radarů patří, kromě zmíněného sledování pohybu a navigace letadel, zejména pozorování přírodních útvarů, např. meteoradary[1] či georadary[2], vojenské použití při sledování nespolupracujících či nepřátelských objektů, ostraha prostorů, měření rychlosti nebo v automobilové dopravě. [1] [2]

[1] Meteoradar je speciálním typem radiolokátoru, určený k detekci srážkové oblačnosti.
[2] Georadar je taktéž speciálním typem radaru. Je určen ke zkoumání a detekci objektů pod úrovní terénu. (Daniels, 2004)



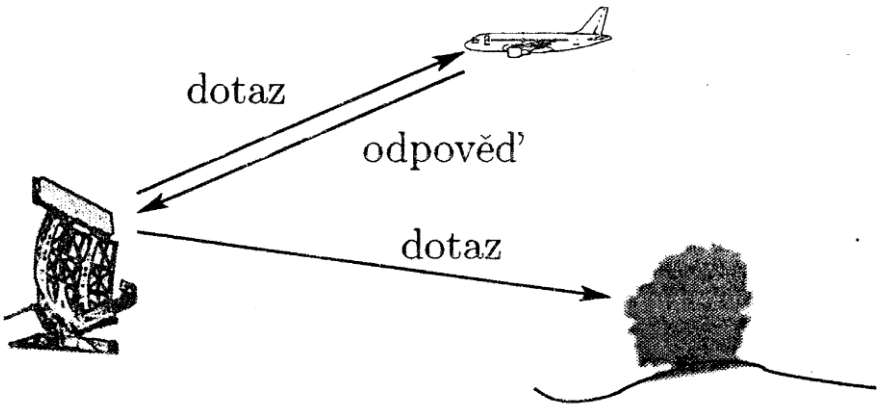
Obrázek 1 – Schéma fungování primárního radaru. Převzato z [1]

1.4.2 Sekundární radar

Sekundární radar, jinak nazývaný také jako dotazovač, stejně jako primární radar, vysílá do svého okolí elektromagnetickou vlnu. Na rozdíl od něj nepřijímá její odraz, nýbrž signál vyslaný odpovídačem umístěným na sledovaném objektu. Tento odpovídač, taktéž označovaný jako transpondér, zachytí signál vyslaný radarem a s předem definovaným zpožděním vyšle dohodnutou odpověď. Vyslaná odpověď zpravidla bývá na jiné frekvenci, než dotaz, a je zachycena a zpracována opět anténou dotazovače. [1]

Určení vzdálenosti a úhlové polohy se provádí na základě stejného principu jako u primárního radaru, tedy vyhodnocením zpoždění odpovědi a směrových vlastností antény. Součástí odpovědi však bývá také datová část, obsahující informace o sledovaném objektu, jimiž může být např. číslo letu, poloha dle GPS, barometrická výška apod. [1]

Sekundární radar je schopen detekovat pouze spolupracující objekty disponující příslušnými odpovídači. Tato nevýhoda je vyvážena delším dosahem a možností získat doplňující informace o sledovaném objektu. Oproti tomu je výhodou sekundárních radarů omezení přijímaných signálů na odpovědi odpovídačů, kterým nekonkurují žádné signály odražené od terénu, mraků či vegetace, viz obrázek 2. Sekundární radary se využívají např. v řízení letového provozu či lodní navigaci. [1]



Obrázek 2 – Schéma fungování sekundárního radaru. Převzato z [1]

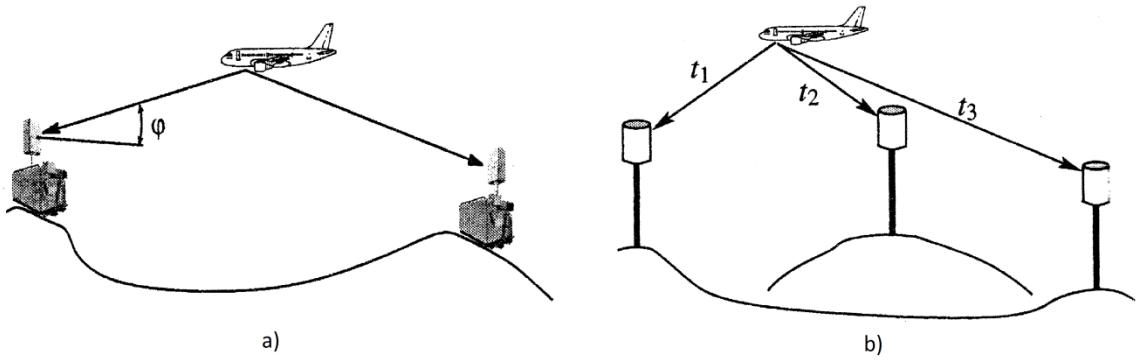
1.4.3 Pasivní radary

Pasivní radary fungují na principu zpracování signálu generovaného jinými objekty. Zdroje tohoto signálu se mohou nacházet buď přímo na sledovaných objektech, nebo mohou být generované externími zdroji a od sledovaného objektu se pouze odrážet. Pasivní radary nezatěžují[3] své okolí elektromagnetickým signálem, protože žádný neemitují. Ze stejného důvodu není pro jejich činnost nutné získat přidělení vysílací frekvence. [1]

1.4.4 Ostatní typy radarů

Kromě primárních, sekundárních a pasivních radarů existuje ještě řada jiných specializovaných typů radarů. Směroměrné systémy se skládají z přijímačů využívajících směrových vlastností antén k určování směru příchodu signálu. Ve větším počtu takových směroměrných senzorů lze zkombinováním získaných údajů určit úplnou polohu sledovaného objektu. Dopplerovské radary využívají k určení polohy a rychlosti sledovaných objektů posuv kmitočtu v jednotlivých okamžicích. Určení polohy dopplerovskými systémy je méně pohotové a podmínkou jeho fungování je vzájemný pohyb radaru a sledovaného objektu. Časoměrné systémy využívají k určování polohy objektu časoměrnou metodu. Jednotlivé přijímače měří časový okamžik příchodu signálu odraženého či vyslaného sledovaným objektem a z rozdílu změřených časů vypočítávají jeho polohu. Výhodami časoměrných systémů je možnost využití malých antén se širokým úhlem záběru, a tím jejich levná konstrukce, vysoká přesnost určení polohy a špatná možnost detekce těchto zařízení nepřátelskými subjekty, protože samy nevyzařují žádnou energii. Jednotlivé antény těchto systému musí však být dostatečně daleko od sebe. Vzdálenost bývá v jednotkách až desítkách kilometrů. Viz obrázek 3. [1]

[3] Elektromagnetické znečištění, EMC

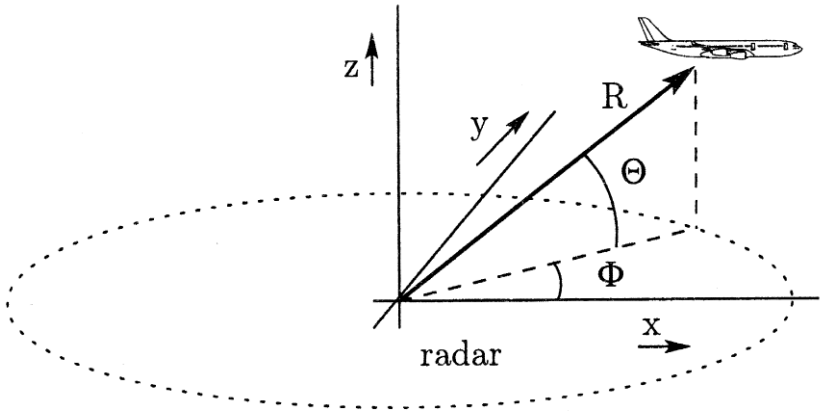


Obrázek 3 – Schéma fungování a) směroměrných systémů a b) časoměrných systémů. Převzato z [1]

Jelikož náplní této práce je zpracování meteorologických srážkových informací pořízených primárním přehledovým radarem, nebude v jejím dalším obsahu věnována pozornost jiným typům radarů. Všechny dále zmíněné informace se budou vztahovat právě k primárním radarům. Informace k ostatním typům radarů lze najít např. v [1].

1.5 Měření parametrů polohy cíle u primárních radarů

Poloha cíle bývá u primárního radaru zpravidla vyjadřována ve sférické soustavě souřadnic s počátkem v místě radaru. Měřenými souřadnicemi tedy jsou šikmá vzdálenost, úhel v ekvatoriální rovině a úhel nad ekvatoriální rovinou. Kromě těchto parametrů primární radary často měří také radiální rychlost. V následujících odstavcích budou nastíněny principy jejich určování. [1]



Obrázek 4 – Znázornění detekovaného objektu ve sférických souřadnicích. Převzato z [1]

1.5.1 Šikmá vzdálenost

U primárního radaru se vzdálenost cíle určuje ze zpoždění mezi vyslaným signálem a přijatým signálem, odraženým od cíle. Měření vzdálenosti může být realizováno několika metodami. Většina radarů s dosahem větším než stovky metrů pracuje v impulzním režimu, tedy vysílá v pravidelných intervalech krátké impulzy elektromagnetické energie.

Mezi těmito impulzy radar určitou dobu nevysílá, ale pouze přijímá odražené signály. Méně často používanými přístupy jsou metoda s kontinuálním vyzařováním a metoda automatického sledování cíle v dálce. [1] [2]

1.5.2 Určování směru cíle

K určování směru, ve kterém se cíl nachází, se u primárních radarů využívá směrových vlastností antény. Směrová, neboli amplitudová, charakteristika antény vyjadřuje závislost amplitudy intenzity elektrického pole na úhlech, obvykle vztažených k nějakému význačnému směru, nejčastěji k maximu anténní charakteristiky. Výkonová charakteristika antény má výrazné maximum, zvané hlavní lalok, a řadu vedlejších maxim, postranních laloků. Hlavní lalok za provozu radaru postupně mění svou polohu v prostoru a prohledává jej. Pohyb laloku je zajištěn buď mechanicky, pohybem celé antény, nebo elektronicky. Toto prohledávání se nazývá snímáním. Vztažným směrem k vyčíslení směrového úhlu zpravidla bývá geografický sever. [1]

1.5.3 Určování rychlosti cíle

Rychlost sledovaného objektu lze určit především výpočtem z již naměřených poloh v závislosti na čase. Změří-li se poloha pohybujícího se bodu ve dvou časových okamžicích, lze jeho rychlost snadno dopočítat. Přijatelnou přesnost poskytne interval mezi měřeními, roven době otočení antény. Další možností měření rychlosti sledovaného objektu je využití Dopplerova jevu, tedy z posunu kmitočtu jím způsobeným. [1]

1.6 Odraz rádiových vln

Při dopadu elektromagnetických vln na jakýkoliv hmotný objekt dochází k jejich rozptylu a kromě dopadajících vln se prostorem začnou šířit i vlny rozptýlené, běžně nazývané odražené[4]. Ve vztahu k primárním radarům lze rozřadit objekty odrážející vyzářený signál, se kterými se setkáváme nejčastěji, do dvou hlavních skupin, kterými jsou:

1. bodové objekty – objekty, jež jsou svými rozměry menší, než je velikost rozlišovací buňky radaru,
2. rozsáhlé objekty – objekty, jejichž rozměry jsou podstatně větší, než je rozlišovací schopnost radaru.

Typická rozlišovací buňka má rozměry přibližně 1° v úhlu (ve vzdálenosti 100km odpovídá 1,7km) a 100m v dálce[5]. [1]

Typickými představiteli bodových objektů, odrážejících signál radaru jsou např. letadla, menší lodě, budovy, vozidla nebo stromy. Rozsáhlé objekty lze dále rozřadit do dvou skupin, a to na objemové a plošné. K tzv. objemovým útvarům řadíme meteorologické útvary, zejména déšť, mraky či mlhu, zatímco plošnými objekty nazýváme především terén a vodní hladinu. [1]

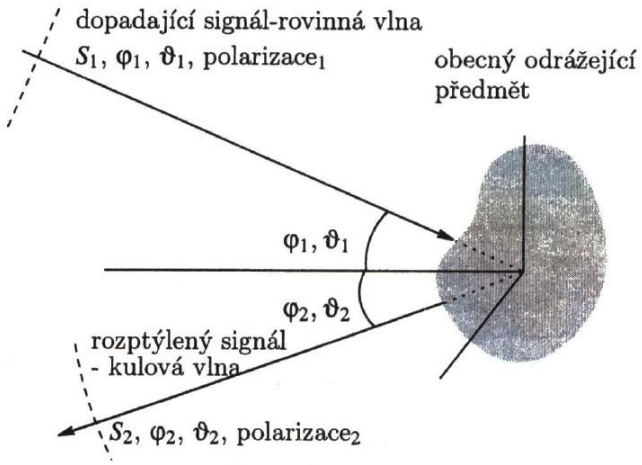
[4] Odraz je jev probíhající na rozhraní dvou prostředí, zatímco k rozptylu vln u nevodivých objektů přispívá také jejich vnitřní část. [1]
[5] Polární systém souřadnic.

1.6.1 Odraz od bodových objektů

Odrazem od bodového objektu se z rovinné vlny stává kulová, kterou lze v okolí objektu aproximovat rovinnou vlnou. Takovou vlnu charakterizujeme pomocí hustoty výkonu dopadající vlny, směru šíření a polarizací dopadající vlny. Hustota výkonu odražené vlny závisí na následujících veličinách:

1. vzdálenosti od odrážejícího objektu,
2. velikosti hustoty výkonu dopadající vlny,
3. směru dopadající vlny a směru pozorování odražené vlny,
4. polarizaci dopadající vlny a polarizaci odražené vlny v místě pozorování,
5. kmitočtu dopadající vlny
6. a na vlastnostech odrážejícího předmětu.

Pokud na objekt dopadá lineárně polarizovaná vlna, odráží se se stejnou lineární polarizací. Pokud však dopadající vlna je kruhově polarizovaná, změní se se změnou směru šíření také smysl její polarizace. Objekty složitější než koule zpravidla díky své nesymetrii a mnohonásobným odrazům odráží vlny v obou ortogonálních polarizacích. Tohoto rozdílu v polarizačních vlastnostech se u radarů využívá k oddělení odrazů od deště a od ostatních objektů. [1]



Obrázek 5 – Odraz vlny od bodového objektu. Převzato z [1]

1.6.2 Odrazy od rozsáhlých útvarů

Typickým zástupcem rozsáhlých objemových útvarů je déšť. Zpravidla pokrývá území o rozloze minimálně několika km^2 a ve vertikálním směru dosahuje až do výšek několik km nad hladinou moře. Síla zachyceného odraženého signálu se pak v jedné rozlišovací buňce radaru rovná součtu sil vln odražených od jednotlivých kapek. Z plošných rozsáhlých útvarů je pro radar důležitý především terén a vodní hladina. [1]

1.7 Identifikace meteoinformací

Jak již bylo zmíněno výše, k rozlišení skutečnosti, zda je zachycený signál odražen bodovým nebo rozsáhlým objektem, je využito polarizačních vlastností rádiových vln. K odlišení těchto informací se u primárních radarů využívá polarizačních prvků vysokofrekvenční obvodů. Tyto prvky filtrují nebo přesměrovávají vlny s ortogonální polarizací. Takto přijímané ortogonálně polarizované vlny tedy mohou být buď potlačeny anebo např. samostatně zpracovány jako pomocný „meteo“ kanál pro zobrazení mraků, deště a dalších meteoútvarů. Tento meteo kanál bude sloužit jako zdroj dat pro analýzu, jíž je tato práce věnována. Podrobněji se této problematice věnuje [1].

2 Jednotné evropské nebe

Projekt jednotného evropského nebe (SES, z anglického Single European Sky) je v současnosti jedním z nejvýznamnějších projektů Evropské komise v oblasti letecké dopravy. Vznikl jako reakce na dramatický růst letového provozu v 80. a 90. letech 20. století a významný nárůst zpoždění na jeden provedený let v letech 1999 - 2000. Hlavním cílem projektu bylo zvýšit kvalitu poskytovaných letových provozních služeb v evropském prostředí. Prioritami bylo zvýšení kapacity a propustnosti vzdušného prostoru při zachování vysoké míry bezpečnosti a maximální kvality poskytovaných služeb. Jednou ze součástí programu SES je nařízení Evropského parlamentu a Rady EU o interoperabilitě ATM[6] systémů s cílem zajistit jejich vzájemnou automatizovanou výměnu dat, prostřednictvím sady protokolů, vytvořené speciálně pro tento účel, viz kapitola 2.2. Organizací zastřešující navrhování, vydávání, standardizaci a podporu projektu SES je EUROCONTROL[7]. Jednou z částí těchto specifikací je i přenos meteorologických informací, jimž bude věnována pozornost v kapitole 2.3. [3] [8]

2.1 Řízení letového provozu

Řízení letového provozu (ATC) patří mezi nejvýznamnější oblasti využití radiolokačního principu a jeho význam roste úměrně s narůstající hustotou letecké dopravy. Jedná se o službu poskytující letadlům pokyny, rady a informace nezbytné k zajištění bezpečného a ekonomického leteckého provozu, přičemž radiolokátory jsou k tomu klíčovým prostředkem. Jejich využití je zřejmé. Pomocí radiolokátorů se sledují pozice letadel, směr, výška a rychlost jejich letu. [4]

2.2 ASTERIX

EUROCONTROL přispívá k podpoře subjektů, podléhajícím mezinárodním regulím, množstvím koordinačních aktivit, ve snaze o společné dosažení legislativních požadavků. Jednou z těchto aktivit je projekt ASTERIX[8]. Jedná se o binární formát předávání zpráv pro potřeby ATM. Jeho smyslem je harmonizace přenosů informací mezi všemi zapojenými systémy. Definuje strukturu přenášených dat od kódování jednotlivých bitů, až po jejich organizaci do větších bloků na aplikační úrovni. [5]

ASTERIX je formátem aplikační a prezentační vrstvy referenčního síťového modelu ISO/OSI[9]. Díky tomu je nezávislý na fyzickém médiu, kterým jsou data přenášena. Pro potřeby přenášení informací vztažených ke konkrétnímu účelu, jsou datové položky seskupovány do kategorií, kterých může být až 256. Definice a sestavování těchto kategorií má na starosti ASTERIX Maintenance Group[10]. Příkladem skupin jednotlivých kategorií jsou např. kategorie 048, určená k přenosu zpráv o cílech identifikovaných přehledovými

[6] ATM - management letového provozu
[7] http://www.eurocontrol.int/
[8] Z angl. **A**ll Purpose **ST**ructured **E**urocontrol Su**R**veillance **I**nformation E**x**change.
[9] Síťový model ISO/OSI, přijatý roku 1984 jako mezinárodní norma ISO 7498.
[10] Více podrobnějších informací je k dispozici na http://www.eurocontrol.int/services/asterix.

systémy, kategorie 034 určená k přenosu servisních zpráv mezi systémy, či kategorie 008, specifikující formát přenášení meteoinformací[11]. [5]

2.3 ASTERIX CAT008

Kategorie 008 (Transmission of Monoradar Derived Weather Information) je stěžejní kategorií pro potřeby této práce. Jejím obsahem je popis přenosu relativně jednoduchých meteorologických obrazů srážkových oblastí s různými úrovněmi intenzity z radarové stanice dále do systémů zpracovávajících tato data. Je platná od prosince 1997.

Kategorie specifikuje následující dva typy zpráv:

1. datové – obsahující elementy, tvořící obraz počasí
2. a kontrolní.

Kontrolní zprávy jsou dvě a jejich úkolem je vymezení platnosti datových zpráv. Informují především o začátku a konci cyklu obnovy informací o stavu srážek. Datové zprávy již nesou samotné informace o srážkovém obrazu, a to v jednom ze tří možných způsobů jejich reprezentace.

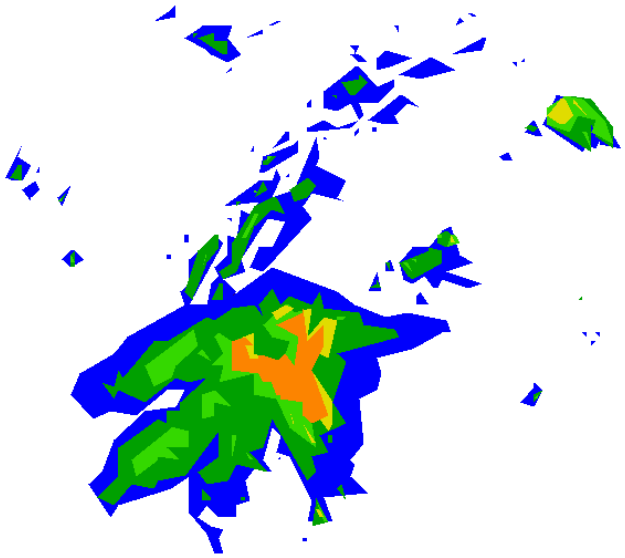
2.3.1 Reprezentace srážkových oblastí

Dle specifikací kategorie 008 je možné srážkové oblasti reprezentovat jednou z následujících tří možností:

1. stínované plochy polárních vektorů – tedy sérií blízko sebe umístěných vektorů, vyjádřených v lokálních polárních souřadnicích,
2. stínované plochy kartézských vektorů – tedy tak, že srážková oblast bude vyjádřena sadami paralelních vektorů v kartézském sytému souřadnic s počátkem buď v místě radaru nebo jiného referenčního bodu, nebo
3. kontury srážkových oblastí – tedy sérií po sobě jdoucích bodů, tvořících konturu uzavřené oblasti.

Poslední zmíněná možnost by měla být použita v případě, že jsou srážkové oblasti vykreslovány jinak, než stínováním, např. vybarvováním různými barvami podle intenzity, jak je vidět na obrázku 6. Dále v případech, kdy je důležitá minimalizace přenášených informací, speciálně pak pokud je na kontury aplikována nějaká redukční technika s efektem snížení přesnosti kontury (pro příklady takových technik viz kapitolu 3.4). [6]

[11] Kompletní seznam včetně specifikací je přístupný online na adrese http://www.eurocontrol.int/services/specifications-documents



Obrázek 6 – Reprezentace srážkových oblastí formou polygonů vybarvených dle intenzity

3 Proces zpracování přijatých dat

Následující části práce se již budou věnovat výhradně meteo kanálu dat z primárního radaru, představenému v kapitole 1.7. Tento kanál se v ATC systémech využívá především z důvodu snahy o zvyšování bezpečnosti letového provozu. Specifikace ASTERIXu zavádí rozlišení srážkových informací do 8 úrovní intenzit. Nejvyšší intenzity již většinou značí špatnou meteorologickou situaci, především přítomnost bouřek, silného deště či větru. Snahou je navést letadla mimo tyto zóny. [6]

Jelikož informace získané z meteo kanálu jsou předávány ve formě skupiny číselných hodnot reprezentujících intenzitu srážek pro odpovídající vzdálenosti v konkrétním úhlu, je před samotnou analýzou obrazu zachycených meteorologických informací zapotřebí surová radarová data interpretovat do zpracovatelné podoby.

Analýzou připravených srážkových dat, tedy převedením obrazu na množinu polygonů reprezentujících jednotlivé srážkové objekty v atmosféře, je myšlen postup aplikování několika po sobě jdoucích kroků, vedoucích k očekávanému výsledku. Navržený postup zpracování se skládá ze tří na sebe navazujících kroků, kterými by měl každý obraz projít. Prvním krokem analýzy je filtrace obrazu, jejímž účelem je především odstranění šumu a rozmazání detailů nepřinášejících žádnou přidanou hodnotu pro výsledek analýzy, avšak způsobujících snížení rychlosti zpracování v následujících krocích. Druhým krokem je prohledání celého obrazu a identifikace obrazců, reprezentujících srážkové objekty. Třetím a posledním krokem analýzy je tzv. zjednodušení nalezených polygonů. Poslední krok je do analýzy zařazen z důvodu odstranění bodů z kontury polygonu takovým způsobem, že významně neovlivní tvar objektu, ale přesto přispěje ke snížení objemu výsledných dat.

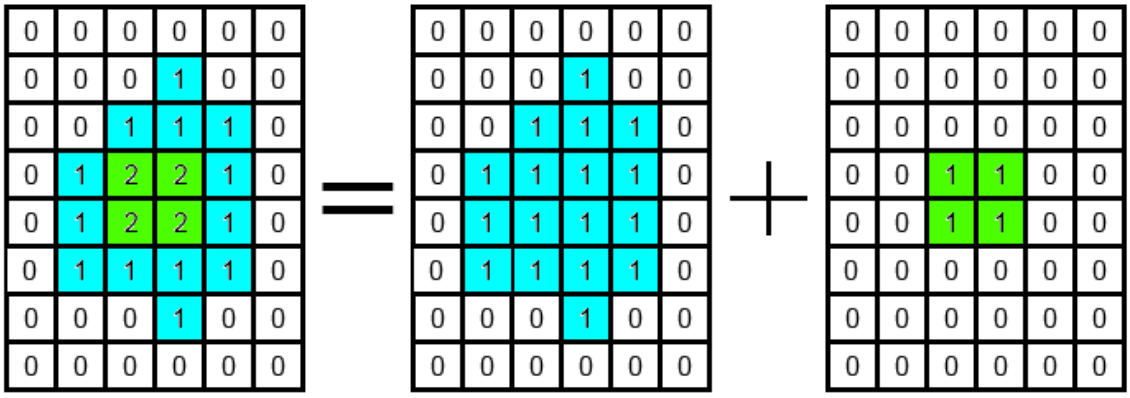
První ani třetí krok nejsou z hlediska specifikací ASTERIX povinné. Zařazení filtrace obrazu je navržena z hlediska možné úspory času při vyhledávání kontur. Oproti tomu třetí krok je zařazen kvůli zvýšení komprese vypuštěním nedůležitých detailů.

3.1 Sběr informací v průběhu otáčky pro zpracování

Pro potřebu analýzy je nezbytné získat ucelený obraz srážkové situace detekované radarem. Na začátku každé otáčky je připraveno sedm čistých rastrů reprezentujících geografickou oblast v dosahu radaru, přičemž každý rastr představuje srážkovou situaci nad sledovaným územím pro danou intenzitu srážek. V okamžiku příchodu úhlového skenu jsou informace v něm obsažené zaznamenány do rastru odpovídající úrovně a všech rastrů nižších úrovní[12].

Na obrázku 7 je graficky znázorněn rozklad příchozích srážkových dat z meteo kanálu, obsahující tři úrovně intenzity srážek (0, 1 a 2). Barevně jsou vyznačeny hledané objekty.

[12] Pro každou srážkovou oblast s intenzitou vyšší než 1, je uvažováno, že obsahuje také srážky o nižších intenzitách.



Obrázek 7 – Rozklad meteodat o třech úrovních intenzity do rastrů

Jelikož většina radarů pracuje s polárním systémem souřadnic, je nutné poskytnutá data přepočítat do kartézského souřadnicového systému, se kterým pracují obrazové rastry. Úhlový sken dává množinu informací o srážkách z konkrétního úhlu θ[13], kvantované po vzdálenostech ρ, viz obrázek 8. Ze známé vzdálenosti, úhlu, přírůstku úhlu $\Delta\theta$ a přírůstku vzdálenosti $\Delta\rho$ lze snadno pouhým přičtením a odečtením dle vztahu (0.1) určit polární souřadnice bodů *A, B, C a D*, ohraničujících danou srážkovou informaci.

$$[\rho_P, \theta_P] = [\rho \pm \Delta\rho, \theta \pm \Delta\theta] \tag{0.1}$$

Pro zakreslení oblasti ohraničené těmito čtyřmi body je však nutné je převést do kartézského souřadnicového systému, se kterým pracuje rastr. Aby to bylo možné je potřeba znát rozměry rastru a maximální dosah radaru. Nejprve je nutné převést vzdálenost bodu v kilometrech na vzdálenost vyjádřenou v pixelech dle vztahu (0.2), kde ρ_{pix} je hledaná vzdálenost v pixelech, ρ je vzdálenost v kilometrech, ρ_{max} je maximální dosah radaru a a je rozměr čtvercového rastru.

$$\rho_{pix} = \rho \cdot \frac{\frac{a}{2}}{\rho_{\max}} \tag{0.2}$$

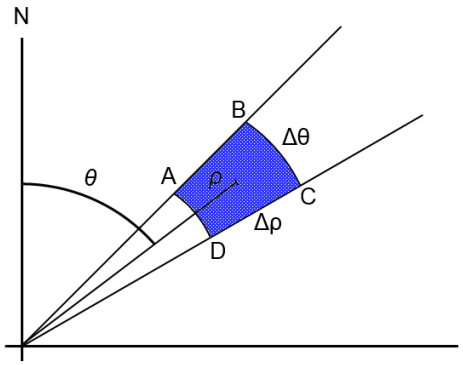
Souřadnice *x, y* jednotlivých bodů pak lze vypočíst podle vztahu (0.3).

$$[x_P, y_P] = [\rho_{pix} \cdot \sin\theta_P, \rho_{pix} \cdot \cos\theta_P] \tag{0.3}$$

Jako dostatečně rychlý a přesný se osvědčil postup zakreslování jednotlivých oblastí do rastru pomocí známých algoritmů, určených k vybarvování plochy polygonů v rastrové grafice, např. paritním vyplňováním (viz [7]). Výsledný obraz ukázkové srážkové situace jedné úrovně intenzity ilustruje obrázek 9. Tento postup je za cenu mírných nepřesností v okrajových částech zakreslených srážkových oblastí vysoce univerzální a nezávislý na

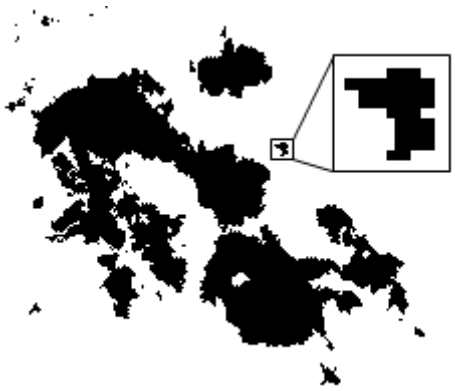
[13] Označení úhlu θ a vzdálenosti ρ převzato z [6]

hustotě zvoleného rastru, či na poměru velikosti bodu rastru vůči rozsahu zakreslované oblasti. Tyto problémy implicitně řeší vybarvovací algoritmus.



Obrázek 8 – srážková informace v úhlu θ a vzdálenosti ρ

Takto je tedy pro analýzu připraveno ke zpracování sedm rastrů pro sedm úrovní. Rastr pro danou úroveň obsahuje jednobitovou informaci o tom, zda daná oblast odpovídající intenzity obsahuje srážky či nikoli. Vzhledem k tomu, že jde pouze o jednobitovou informaci pro každou intenzitu, není potřeba na původní obraz aplikovat metody prahování.



Obrázek 9 – Příklad zaznamenaného vstupu jedné úrovně intenzity do analýzy

Výstup této činnosti je vstupem pro samotnou analýzu, nikoli její součástí, a proto jí dále v této práci nebude věnována zvýšená pozornost.

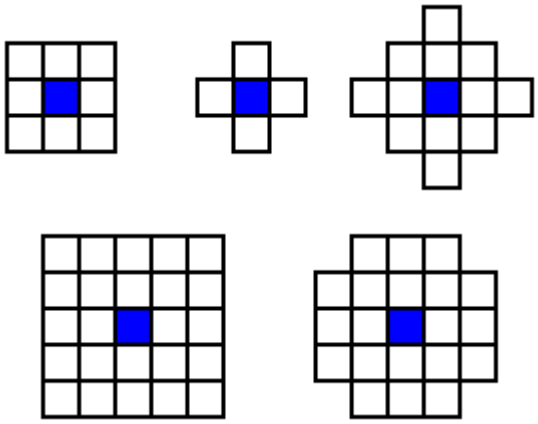
3.2 Filtrování obrazu

Jakmile jsou data připravena do rastrové podoby, přichází první krok analýzy. Informace v jednotlivých rastrech jsou zpravidla velmi nehomogenního charakteru, viz obrázek 10 vlevo. To znamená, že obsahují velké množství drobných nespojitých oblastí, které nemají pro potřeby řízení letového provozu žádný význam a vedly by k vyšším nárokům na zpracovávací algoritmy z druhého kroku analýzy, vyplývajícím z principu jejich činnosti. Proto se jeví žádoucí před dalším zpracováním data filtrovat. Filtrováním je tedy myšleno

především odstranění šumu v podobě drobných oblastí a odstranění složitosti obrysů větších oblastí, které i přesto, že jsou spojité, mohou mít hranice velice roztřepeného charakteru. Taková členitost hranic objektu by také mohla vést k nežádoucímu zvýšení náročnosti výpočtu bez vlivu na hodnotnost výsledné informace. Na obrázku 10 je ilustrován cílený efekt předzpracování obrazu filtrováním.

Obrázek 10 – Očekávaný efekt filtru (vlevo situace před filtrací, vpravo po filtraci)

Nejběžnější metodou filtrování obrazu je aplikace konvoluční masky na původní obraz. Prakticky se jedná o úpravu intenzity obrazových bodů na základě intenzity jasu okolních bodů, přičemž okolí může mít různý tvar, viz příklady okolí na obrázku 11.



Obrázek 11 – Příklady různých typů symetrických okolí bodu. Převzato z [9]

Filtry se dělí na lineární a nelineární. Lineární lze dále rozdělit na filtry typu dolní propust a horní propust. Příkladem nelineárních filtrů jsou filtry typu minimum, medián či maximum. [9]

Při aplikaci lineárních filtrů se výsledná hodnota bodu rovná součtu součinů intenzit okolních bodů a příslušných váhových koeficientů z matice váhových koeficientů. Filtry typu dolní propust slouží především k odstranění vysokých prostorových frekvencí intenzit, díky čemuž dochází k potlačení šumu, ale také detailů v obraze. Používají se nejčastěji k odstraňování zrnitosti v obrazech. Příklady matic váhových koeficientů

o rozměrech 3x3 jsou matice průměrovací masky ve vzorci (0.4) a dvě ukázky různých matic masky s Gaussovským rozdělením váhových koeficientů ve vzorci (0.5). Typickým znakem filtrů typu dolní propust je, že součet váhových koeficientů je roven 1. [9]

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \tag{0.4}$$

$$\begin{bmatrix} \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{5} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \end{bmatrix} \qquad \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \tag{0.5}$$

Lineární filtry typu horní propust umožňují zvýraznění detailů v obraze, avšak obvykle za cenu zvýraznění šumu. Charakteristickým rysem filtrů typu horní propust je skutečnost, že součet všech koeficientů je roven 0. Příkladem matice váhových koeficientů typu horní propust je vzorec (0.6). [9]

$$\begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix} \tag{0.6}$$

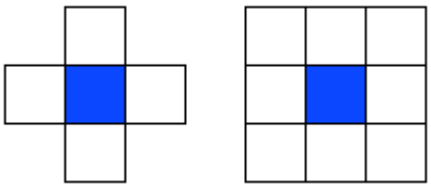
Nelineární filtry se od lineárních liší tím, že nepočítají novou intenzitu upravovaného bodu, ale vybírají a dosadí na jeho místo vybranou hodnotu z jeho okolí. Jejich výhodou je, že na rozdíl od filtrů lineárních nepřidávají do obrazu žádnou novou hodnotu intenzity, což odstraní potřebu provést prahování, které by bylo nutno použít u lineárních filtrů pro opětovné převedení obrazu na jednobitovou hloubku. Filtry minimum a maximum, taktéž nazývané jako eroze a dilatace, vybírají ze svého okolí minimální, resp. maximální hodnotu, kterou dosadí do upravovaného bodu. Pro použití v dvoubarevném rastru tedy nejsou vhodné. Filtr typu medián dosadí do upravovaného bodu hodnotu, která se v jeho okolí vyskytuje nejčastěji. Tento filtr je velmi účinný pro potlačení šumu a lze jej

s výhodou využít pro vyhlazování zrnitosti v obraze. Jeho praktickou nevýhodou (avšak pro naše potřeby spíše výhodou) je, že ohlazuje hrany objektů, čímž mění jejich tvary. [9]

3.3 Hledání kontur objektů

Po odfiltrování nežádoucích informací jsou data připravena pro identifikaci srážkových objektů v obraze a jejich popisu posloupností bodů, tvořících jejich kontury. Jde o nejdůležitější část analýzy, jejímž cílem je identifikovat objekty, které se v něm nachází.

Aby bylo možné identifikovat objekty v obraze, je nutné najít skupiny obarvených bodů, které jsou navzájem „spojené“. Jinými slovy, objekty v obraze jsou skupiny obarvených bodů, které jsou navzájem sousedné. Sousednosti existují dva typy, a to 4-sousednost a 8-sousednost. Sousednými body v rámci 4-sousednosti jsou body nacházející se vedle uvažovaného bodu ve vertikálním a horizontálním směru, tedy čtyři body. V rámci 8-sousednosti se jako sousedné body uvažují body nacházející se těsně u uvažovaného bodu ve vertikálním, horizontálním a také diagonálním směru, celkem tedy osm bodů, viz obrázek 12. Podle těchto kritérií lze rozřadit objekty v obraze na 4-sousedné nebo 8-sousedné podle toho, kterými sousedními body jsou spojené. Z výše zmíněného také vyplývá, že všechny 4-sousedné objekty jsou zároveň 8-sousednými, ale ne všechny 8-sousedné objekty jsou také 4-sousedné. [10]



Obrázek 12 – Sousedství bodu (vlevo sousedné body pro 4-sousednost, vpravo sousedné body pro 8-sousednost)

Hledání kontur objektů je technika využívaná v digitálním zpracování obrazů za účelem nalezení jejich hranic. Hranicí se rozumí posloupnost bodů definující okraje daného obrazce. Hranice mohou být dvojího druhu, v závislosti na tom, zda uvažujeme návaznost bodů v rámci 4-sousednosti či 8-sousednosti, viz obrázek 12. Pro účel zpracování srážkových objektů není podstatné, zda nalezená hranice bude 4-sousedná, nebo 8-sousedná, protože se předpokládá následné zjednodušení kontury, čímž dojde téměř k celkovému odstranění sousednosti v rámci obrysových bodů. Proto je možné do porovnání zahrnout algoritmy pracující jak s čtyřbodovým okolím bodu, tak s osmibodovým okolím bodu. Důležitost tohoto druhu zpracování obrazu spočívá ve značném snížení počtu bodů, potřebných k popisu obrazce oproti nezpracovaným datům, a k odstranění oblastí obrazu bez obrazců. [10]

V následujících podkapitolách bude představeno a vysvětleno pět nejznámějších algoritmů využívaných k identifikaci kontur objektů. Jedná se o algoritmy Marching Squares, Square Tracing Algorithm, Radial Sweep, Moore-Neighbor Tracing a Theo Pavlidis‘ Algorithm.

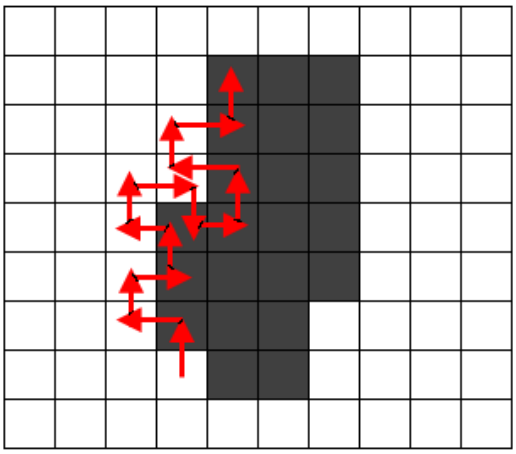
3.3.1 Square Tracing Algorithm

Myšlenka tohoto algoritmu je ve své podstatě velice jednoduchá. Je to z části dáno také tím, že tento algoritmus byl jedním z prvních algoritmů, určených k identifikování kontur objektů v dvoubarevném obraze. [10]

Algoritmus začíná nalezením prvního bodu objektu, což může být uskutečněno mnoha způsoby, např. průchodem obrazu od levého dolního rohu postupně sloupec po sloupci směrem nahoru, dokud se nenarazí na první obarvený bod. Takto nalezený bod je označen jako počátek kontury a dále již nastupuje algoritmus samotný. Důležitým aspektem při zpracování algoritmu je respektování směru, jakým se na daný bod dostane. Postup algoritmu lze popsat pomocí následujících kroků:

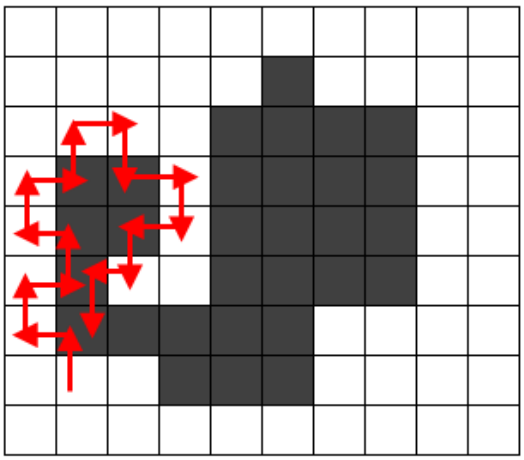
1. jestliže je aktuální bod obarvený, pokračovat přesunem na bod vlevo,
2. jestliže aktuální bod obarvený není, pokračovat přesunem na bod vpravo,
3. ukončit algoritmus při vstupu na počáteční bod.

Obrysem objektu pak je posloupnost obarvených bodů, jimiž algoritmus prošel. Průchod algoritmu ilustruje obrázek 13.



Obrázek 13 – Příklad průchodu algoritmu Square Tracing Algorithm ukázkovým objektem

Avšak tento postup má vážnou slabinu, kterou je způsob ukončení algoritmu. Velmi snadno může nastat situace, kdy algoritmus podruhé vstoupí na počáteční bod a algoritmus skončí i přesto, že ještě nenašel kompletní obrys objektu. Situaci kdy k takovému stavu dojde, ukazuje obrázek 14. [10]



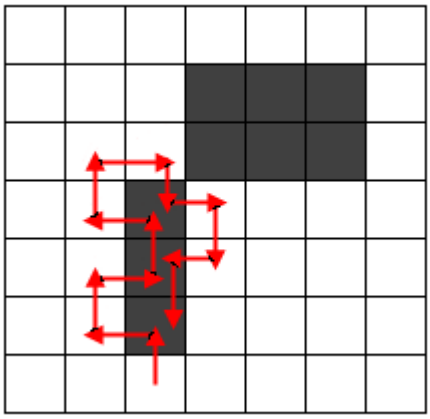
Obrázek 14 – Situace, ve které dojde k předčasnému ukončení algoritmu Square Tracing Algorithm

Jako vylepšení tohoto algoritmu se nabízejí dvě možnosti úpravy kritéria ukončujícího hledání, jimiž jsou:

1. ukončení algoritmu poté, co počáteční bod navštíví n-krát (pro n > 1), nebo
2. ukončení při dosažení počátečního bodu podruhé ze stejného směru, jakým se na něj vstoupilo na začátku.

Druhé zmíněné kritérium navrhnul Jacob Eliosoff, a proto se po něm nazývá Jacobovo ukončující kritérium[14]. [10]

Další komplikací pro použití tohoto algoritmu je uvažování objektů jako 8-sousedných, které nejsou zároveň 4-sousedné. V takovém případě algoritmus vždy selže. Naopak pokud jsou uvažované objekty 4-sousedné, lze jednoduše dokázat, že algoritmus vždy nalezne správnou konturu objektu, viz [10].

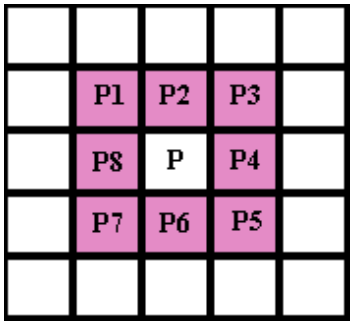


Obrázek 15 – Selhání algoritmu při hledání kontury 8-sousedného objektu

[14] Z angl. Jacob's stopping criterion.

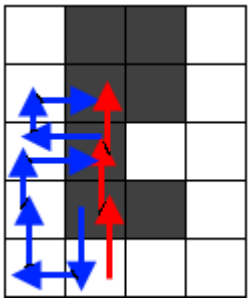
3.3.2 Moore-Neighbor Tracing

Název algoritmu je odvozen od Moorova okolí bodu, které je tvořeno osmi body sousedícími ortogonálně a diagonálně s daným bodem, viz obrázek 16.



Obrázek 16 – Moorovo okolí bodu P tvoří body P1, P2, P3, P4, P5, P6, P7 a P8

Stejně jako u algoritmu z předchozí kapitoly je nutné nějakým způsobem identifikovat počáteční bod objektu. Podstatou samotného algoritmu je potom prohledávání Moorova okolí nalezeného obarveného bodu. Bodem, kterým začne prohledávání okolí, bude poslední navštívený neobarvený bod. Na směru prohledávání okolních bodů nezáleží, dokud zůstane během činnosti algoritmu nezměněn. Činnost algoritmu vystihuje obrázek 17. Modré šipky představují průchod Moorovým okolím, zatímco červené šipky představují samotný posun algoritmu napříč body, jejichž okolí je prohledáváno. Posloupnost těchto bodů také tvoří výslednou konturu. [10]



Obrázek 17 – Ilustrace činnosti algoritmu Moore-Neighbor Tracing

Algoritmus se skládá z následujících kroků:

1. nalezení počátečního bodu *p* kontury objektu,
2. označení počátečního bodu jako aktuálního zpracovávaného bodu *c*, tedy $c = p$,
3. návrat na poslední navštívený neobarvený bod, tento bod náleží do *M(c)*, tedy Moorova okolí bodu *c*,
4. průchod body z *M(c)* ve stanoveném směru,
   4.1. pokud *M(c)* neobsahuje obarvený bod, jedná se o osamocený bod a algoritmus končí bodem 5.
   4.2. V případě nalezení obarveného bodu *b*, se tento bod stává aktuálním bodem, tedy $c = b$. Opakuje se postup od bodu 3.
5. Algoritmus končí ve chvíli kdy $c = p$.

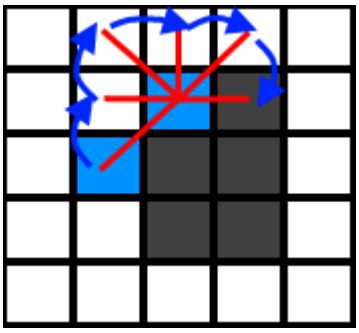
Moore-Neighbor Tracing algoritmus je stejně náchylný na ukončující kritérium, jako předchozí algoritmus, a i řešení tohoto problému je identické jako v případě Square Tracing Algorithm. Na rozdíl od něj však odstraňuje problém s 8-sousedstvím, protože Moorovo okolí pokrývá všech osm sousedních bodů. Algoritmus je tedy při zvolení vhodného ukončujícího kritéria vždy konečný a nalezne správnou konturu objektu, viz [10].

3.3.3 Radial Sweep

Algoritmus Radial Sweep je svým principem velice podobný předchozímu algoritmu, neboť také prohledává Moorovo okolí bodu, ovšem mírně odlišným způsobem. První a druhý bod obrysu jsou nalezeny stejně jako u předchozího algoritmu, změna tedy nastává až od hledání třetího bodu. V tuto chvíli jsou poslední dva nalezené obarvené body spojeny imaginární spojnicí a další bod je hledán „rotováním“ touto spojnicí okolo posledního nalezeného bodu Moorovým okolím, dokud nenarazí na další obarvený bod. Poté se vytvoří nová spojnice a postup se opakuje. K ukončení činnosti algoritmu dojde ve chvíli, kdy algoritmus podruhé navštíví počáteční bod. [10]

Postup algoritmu je následující:

1. nalezení počátečního bodu *p* kontury objektu,
2. nalezení druhého bodu kontury průchodem *M(p)*,
   2.1. pokud takový bod neexistuje, jedná se o osamocený pixel a algoritmus končí,
   2.2. v opačném případě je nalezený bod označen jako aktuální bod *c*.
3. Vytvoří se imaginární spojnice mezi bodem *c* a předchozím nalezeným bodem kontury.
4. Touto spojnicí se rotuje takovým způsobem, že jeden její konec je stále v bodu *c*, zatímco její druhý konec prochází *M(c)*, dokud nenarazí na obarvený bod. Tento je pak označen jako nový aktuální bod *c*.
5. Algoritmus končí ve chvíli kdy $c = p$.



Obrázek 18 – Ilustrace principu hledání kontury objektu algoritmem Radial Sweep

Toto kritérium je však stejně jako u předchozích algoritmů nespolehlivé, neboť pro spoustu objektů dojde k předčasnému ukončení. K ukončení nelze využít ani Jacobovo kritérium, protože koncepce algoritmu Radial Sweep nepracuje se směrem, odkud byl bod navštíven. Z tohoto důvodu bylo zavedeno nové ukončovací kritérium. Jeho podstatou je tvorba posloupnosti nalezených obarvených bodů a ukončení činnosti v situaci, kdy dojde k navštívení dvou po sobě jdoucích bodů ve stejném pořadí, v jakém již jednou byly navštíveny. Avšak tento postup vyžaduje při vstupu na každý další nalezený bod zkontrolovat celou aktuálně nalezenou konturu, čímž významně snižuje efektivitu algoritmu. [10]

3.3.4 Theo Pavlidis‘ Algorithm

Algoritmus je nazván jménem svého stvořitele, tedy Thea Pavildise. Jedná se o jeden z nejnovějších algoritmů určených k hledání kontur objektu v dvoubarevném rastru. Poprvé byl publikován roku 1982. [10]

Před vysvětlením principu algoritmu je vhodné zavést označení směrů orientace, např. Sever, Jih, Východ a Západ. Pro použití prohledávání prostoru proti směru hodinových ručiček je zapotřebí mírně upravit způsob hledání počátečního bodu, a to tak, že hledání provedeme z levého horního bodu zleva doprava, řádek po řádku. Tímto je zaručeno, že počáteční obarvený bod nemá ve směru Sever a Západ obarvené sousedy (toto je podmínkou pro případ procházení proti směru hodinových ručiček a při obrácení směru prohledávání je nutno upravit i tato pravidla) a aktuálním směrem orientace je Východ. Nalezení dalšího obarveného bodu probíhá ve třech krocích, jimiž jsou:

1. kontrola tří sousedních bodů *P1*, *P2* a *P3* v nastaveném směru, přičemž:

   1.1. pokud je bod *P1* obarvený, stává se novým výchozím bodem a aktuální směr je změněn na levý-kolmý k předchozímu (ze směru Východ se stane směr Sever atd.), kroky 1.2 a 1.3 se přeskakují.

   1.2. Pokud je bod *P2* obarvený, stává se novým výchozím bodem a směr zůstává nezměněn, bod 1.3 se přeskakuje.

   1.3. Pokud je bod *P3* obarvený, stává se novým výchozím bodem a směr zůstává nezměněn.

2. Vyhodnocení nastalého stavu:

   2.1. pokud je jeden z bodů *P1*, *P2* nebo *P3* obarvený:

      2.1.1. Vynuluj počet rotací a pixel zaznamenej jako hraniční.

      2.1.2. Pokud je obarvený bod počátečním bodem, zkontrolují se jeho přilehlé body podobně jako v kroku 1, ale body nenastavují jako výchozí a nemění se směr. Místo toho je přidána kontrola, zda byly přilehlé body již označeny jako hraniční, a vyhodnotí se jejich stav dle kroku 2.

      2.1.3. Pokud byla přidána kontrola na označení bodů jako hraničních, zkontroluje se, zda již byl jeden z přilehlých bodů počátečního bodu označen jako hraniční. Pokud ano, algoritmus končí.

      2.1.4. Pokračuje se opět krokem 1.

   2.2. Pokud žádný z bodů *P1*, *P2* a *P3* není obarvený:

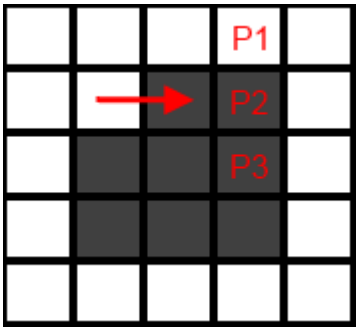
      2.2.1. pokud je počet rotací menší než tři, provede se rotace aktuálního směru po směru hodinových ručiček (tj. pokud byl směr Východ, směr po rotaci bude Jih).

      2.2.2. Pokud je počet provedených rotací větší než tři, znamená to, že tento bod je v rastru osamocený a algoritmus končí.

3. Ukončení algoritmu proběhne pokud:

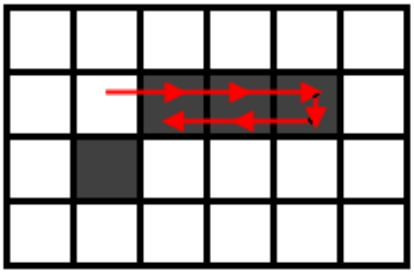
   3.1. se v žádném směru nenachází další obarvený bod, nebo

   3.2. pokud byl navštívený bod již jednou označen jako hraniční v aktuálně nastaveném směru.



Obrázek 19 – Znázorňuje pozici vyhodnocovaných bodů P1, P2 a P3 vzhledem ke směru vstupu na obarvený bod

Ačkoli se tento algoritmus může zdát nejkomplexnější, existuje spousta 8-sousedných vzorů, které nejsou zároveň 4-sousedné, na kterých selže: příklad takového objektu se nachází na obrázku 20. [10]

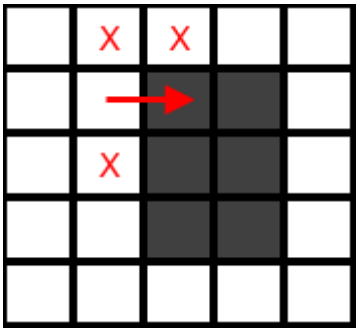


Obrázek 20 – 8-sousedný objekt, pro který Theo Pavlidis algoritmus selže

K vyřešení tohoto problému existují dvě možnosti:

1. ukončení algoritmu po navštívení počátečního kritéria nikoli podruhé, ale až potřetí či počtvrté, nebo

2. vyřešením zdroje tohoto problému, tedy volbě počátečního bodu. Aby bylo zajištěno přesné nalezení kontury, je třeba zvolit takový obarvený bod, který má z pohledu vstupu body vlevo, vlevo za a vpravo za neobarvené, tak, jak to znázorňuje obrázek 21.



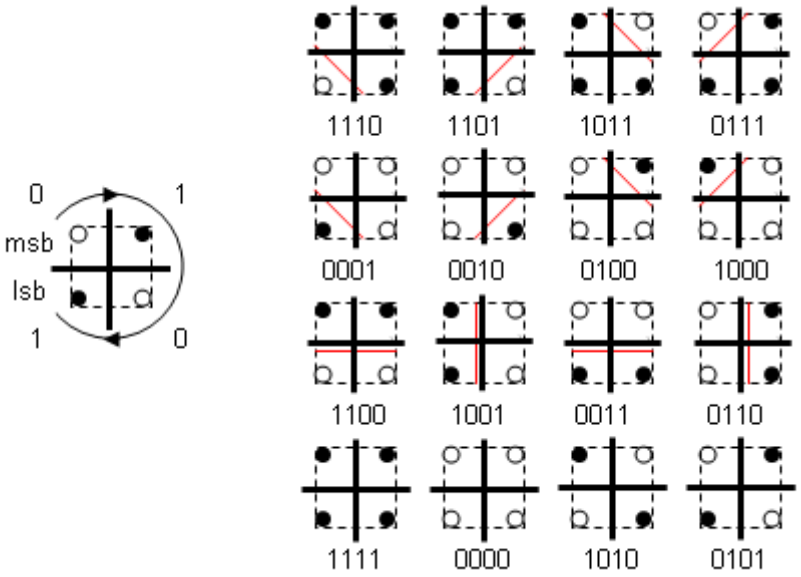
Obrázek 21 – Příklad ideální výchozí pozice pro algoritmus Theo Pavlidis

Bohužel nalezení takového bodu může být ve spoustě obrazců náročné, nebo takový bod vůbec nemusí existovat. V takovém případě musí být použita druhá metoda, tedy ukončení algoritmu po navštívení počátečního bodu potřetí. [10]

3.3.5 Marching Squares

Marching Squares algoritmus je posledním a také nejkomplexnějším algoritmem určeným k hledání obrysů objektu v dvoubarevném obraze, jemuž bude věnována pozornost. Ke své práci potřebuje předem připravenou tabulku možných kombinací barevnosti oblasti bodů o rozměru 2x2, tedy 16 kombinací, viz obrázek 22 (kombinace tvořená čtyřmi neobarvenými body by z principu sledování obrysu neměla být nikdy použita). Algoritmus potom tvoří následující kroky. [11]

1. Nalezení bodu na hranici objektu, např. prohledáváním hrubou silou bod za bodem, řádek po řádku.
2. V dalším kroku uvažujeme oblast o rozměru 2x2 body, jejímž levým horním bodem je bod nalezený v kroku 1.
3. Takto vytvořená oblast je ohodnocena hodnotou 0-15, a to podle toho, kolik obarvených bodů obsahuje a na jakých pozicích (viz Obrázek 22).
4. Na základě hodnoty vypočtené hodnoty z kroku 3 lze určit směr, jakým kontura pokračuje do okolních bodů, a to nahlédnutím do předpřipravené tabulky kombinací.
5. Posledním krokem je posun na nový bod, směrem zjištěným v předchozím kroku a pokračováním na krok 2. Tento postup opakujeme, dokud se nevrátíme podruhé na výchozí bod.



Obrázek 22 – Vlevo způsob ohodnocení matice bodů, vpravo množina všech možných kombinací bodů v matici. Převzato z [11]

3.4 Zjednodušování kontur

Posledním krokem analýzy je tzv. zjednodušování nalezených objektů. V tuto chvíli se mohou identifikované kontury skládat ze stovek až tisíců bodů, tvořících jejich hranice. Tento krok je důležitý pro další kompresi dat určených k přenosu mezi systémy. Cílem zjednodušení těchto kontur je dostatečně přesné vystižení tvaru objektu pomocí minimálního počtu bodů. Takzvané zjednodušující algoritmy jsou podmnožinou generalizačních algoritmů a využívají se zejména v kartografii. V následujících podkapitolách budou představeny vybrané zjednodušovací algoritmy. Byly vybrány podle jejich teoretické účinnosti a přesnosti výsledných tvarů. Podobných algoritmů existuje velké množství, avšak spousta z nich je jen obměnou vyjmenovaných algoritmů, bez přínosu nové heuristiky či výrazné změny principu. Více informací o této problematice v [12], [13] a [15].

3.4.1 Vypuštění každého x-tého bodu linie

Vypuštění, resp. ponechání každého x-tého z N bodů linie je nejjednodušším zmíněným zjednodušovacím algoritmem, kde x je volitelným parametrem. Jak už název napovídá, jedná se o pouhé vypuštění, resp. ponechání bodu z posloupnosti tvořící celou linii, viz obrázek 23. Složitost algoritmu je $O\left(\frac{N}{x}\right)$. [15]

Obrázek 23 – Vypuštění každého druhého bodu linie

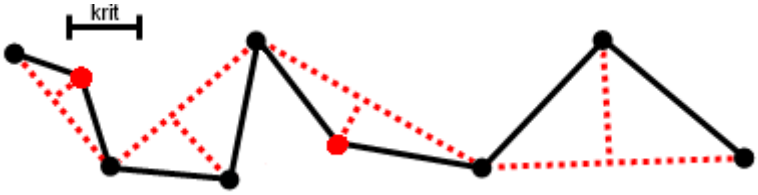
Výhodou tohoto algoritmu je jednoduchost implementace a rychlost zpracování, nevýhodou je pozbytí jakékoli formy heuristiky. Výsledná linie tak může výrazným způsobem zdeformovat původní tvar.

3.4.2 Vzdálenost bodu od strany

Eliminace bodů na základě vzdálenosti bodu od strany je také velmi jednoduchým algoritmem. Jeho podstata spočívá ve zpracování trojic po sobě jdoucích bodů, výpočtu kolmé vzdálenosti v prostředního bodu z trojice od imaginární spojnice dvou okolních bodů podle vztahu (0.7) a v porovnání se stanoveným testovacím kritériem. Pokud je vypočtená vzdálenost kratší než zvolené kritérium, bod je odstraněn. Algoritmus i přes svou jednoduchost dosahuje velmi dobrých výsledků. [15]

$$v = \frac{|ax_a + by_a + c|}{\sqrt{a^2 + b^2}} \tag{0.7}$$

Rozhodovací kritérium je volitelným parametrem algoritmu a bude závislé na požadované přesnosti a velikosti vzdáleností bodů v linii. Výpočetní složitost tohoto algoritmu pro n vrcholů je přibližně $O\left(\frac{n}{3}\right)$.



Obrázek 24 – Vzdálenost bodu od spojnice porovnávaná s testovacím kritériem – vypuštěn bude pouze červeně zvýrazněný bod

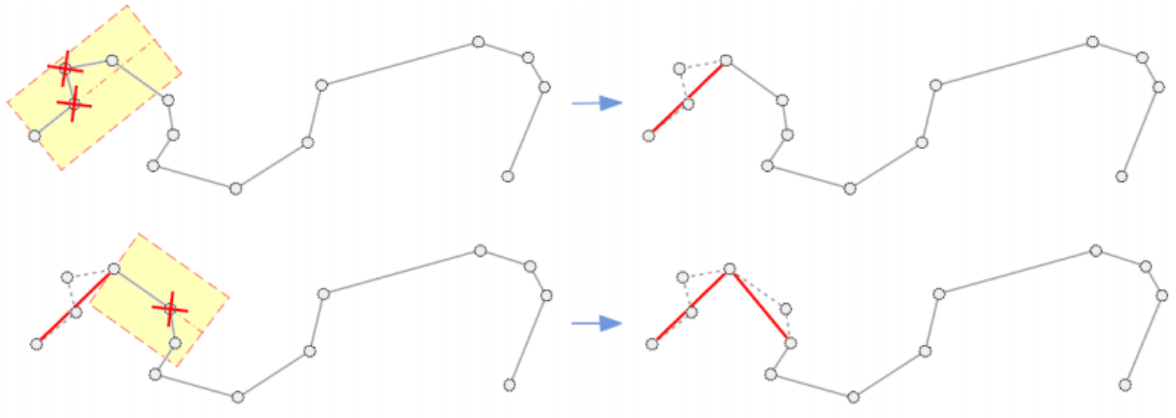
Algoritmus v krocích:

1. nastavení vhodného kritéria *krit*; nastavení prvních tří bodů linie jako *P1*, *P2*, *P3*,
2. spočítání vzdálenosti *v* bodu *P2* od přímky určené body *P1* a *P3*,
3. porovnání vypočtené vzdálenosti *v* s kritériem *krit*,
   3.1. pokud je vzdálenost *v* menší než kritérium *krit*, použijí se ve zjednodušené linii pouze body *P1* a *P3*,
   3.2. jinak se použijí všechny tři body P.
4. Pokud v původní linii jsou za bodem P3 alespoň dva body,
   4.1. nastaví se body P tak, že P1 = P3 a P2 a P3 budou dva následující body a pokračuje se návratem na krok 2,
   4.2. Jinak algoritmus končí

3.4.3 Reumann-Witkamův algoritmus

Reumann-Witkamův algoritmus je prvním zástupcem tzv. rozšířených lokálních algoritmů. To znamená, že neposuzuje vrcholy lomené čáry pouze s ohledem na své sousední vrcholy, ale pracuje s větší množinou bodů a snaží se zohlednit skutečný tvar linie. [14]

Podstatou tohoto algoritmu je opakovaná tvorba koridoru o zvolené šířce *h* rovnoběžně se spojnicí dvou po sobě jdoucích vrcholů linie tak, že spojnice vytvoří jeho osu. Poté se najde průsečík jedné z hranic koridoru s nějakým segmentem linie, který je nejblíže bodům, jejichž spojnice tvoří osu koridoru. Bod takto nalezené linie, nacházející se uvnitř koridoru se stává platným bodem zjednodušené linie a zároveň novým výchozím bodem pro tvorbu dalšího koridoru. Body, které se nacházely mezi tímto bodem a předchozím počátkem koridoru, jsou odstraněny. Postup algoritmu ilustruje obrázek 25.



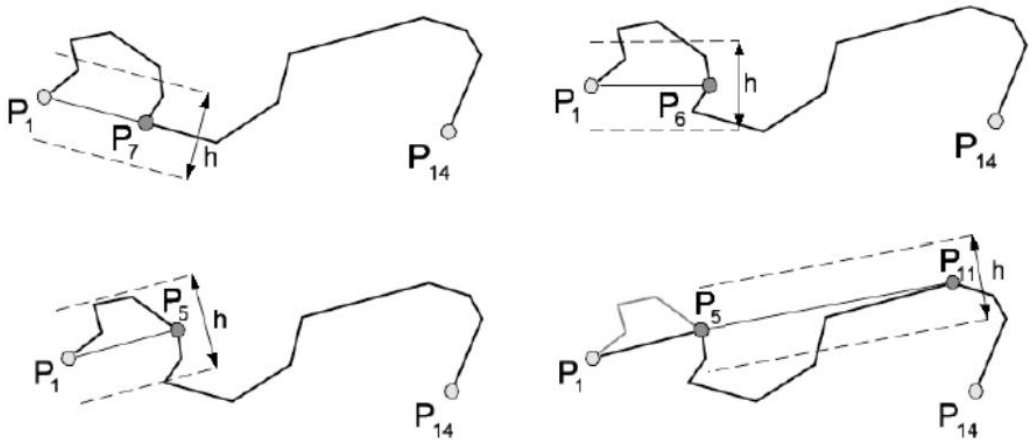
Obrázek 25 – Postup Reumann-Witkamova algoritmu. Převzato z [12]

Algoritmus v krocích: [14]

1. nastavení požadované přesnosti nastavením šířky koridoru *h*,
2. nastavení P_i = první bod linie,
3. vytvoření imaginárního koridoru ve vzdálenosti h/2 od spojnice bodů P_i a P_{i+1},
4. postupným prohledáváním se najde první průsečík hranic koridoru s nějakým segmentem původní linie ohraničeným body S1 a S2,
   - 4.1. pokud je nalezen, jsou odstraněny všechny body mezi Pi a S1, jsou nastaveny body $P_i = S_1$ a $P_{i+1} = S_2$ a pokračuje se krokem 3.
   - 4.2. Pokud není nalezen, jsou odstraněny všechny body mezi Pi a posledním bodem linie a algoritmus končí.

3.4.4 Langův algoritmus

Langův algoritmus, stejně jako Reumann-Witkamův algoritmus, pracuje s koridorem o šířce *h*, ale na rozdíl od něj není tvořen rovnoběžně se stranou lomené čáry, nýbrž rovnoběžně se spojnicí bodů P_i a P_{i+n}, kde *n* je parametr, udávající počet bodů, se kterými se v jednom kroku pracuje. Pokud se alespoň jeden z bodů P_{i+1} až P_{i+n-1} nachází mimo koridor, zůstanou všechny body beze změny a pokračuje se vytvořením nového koridoru, rovnoběžného se spojnicí bodů P_i a P_{i+n-1} a opět se kontroluje, zda nějaký z bodů P_{i+1} a P_{i+n-2} neleží mimo koridor. Takto se pokračuje až do situace, kdy jsou všechny kontrolované body uvnitř koridoru, nebo dokud nedojdou možnosti pro vytváření spojnice bodů. V situaci, kdy se všechny zpracovávané body nachází v koridoru, dojde kromě prvního a posledního k jejich odstranění a z posledního bodu se stává počátek pro vytvoření nového koridoru pro další kroku algoritmu. Postup algoritmu je naznačen na obrázku 26.



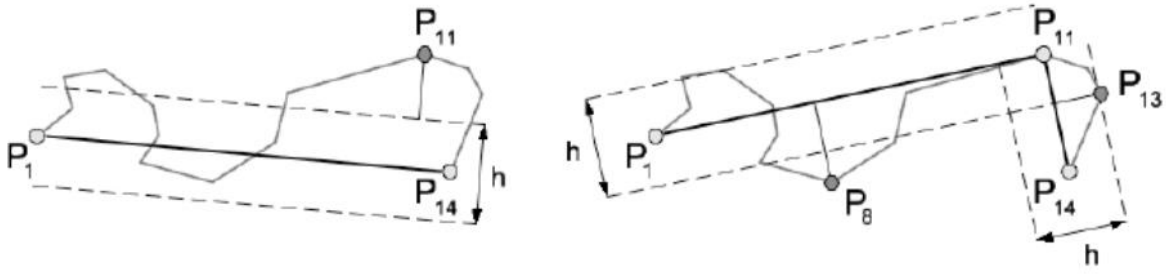
Obrázek 26 – Langův algoritmus pro n=6. Převzato z [12]

Algoritmus v krocích: [14]

1. nastavení požadované přesnosti nastavením šířky koridoru *h*, nastavení výhledového parametru *n*; *i* = 1,
2. nastavení P_i = první bod linie; index $j = i+n$,
3. vytvoření imaginárního koridoru ve vzdálenosti h/2 od spojnice bodů P_i a P_j,
4. prohledáváním od bodů P_{i+1} až P_{i+n-1} se zjišťuje, zda je alespoň jeden z prohledávaných bodů mimo hranice koridoru,
   4.1. pokud takový bod existuje, nastaví se $j = j - 1$, přechod na krok 3.
   4.2. Pokud takový bod neexistuje, jsou odstraněny všechny body mezi P_i a P_j, $i = j$ a pokračuje se krokem 2.
5. Algoritmus končí dosažením posledního bodu linie.

3.4.5 Douglas-Peuckerův algoritmus

Poslední rozebíraný algoritmus patří k nejznámějším a nejpoužívanějším vůbec. Stejně jako předchozí dva algoritmy pracuje s koridorem, ale na rozdíl od nich postupuje opačně. V prvním kroku je celá lomená čára nahrazena spojnicí prvního a posledního bodu, kolem níž se vytvoří zmíněný koridor. Poté se najde bod v největší kolmé vzdálenosti od spojnice a testuje se, zda spadá do koridoru. Pokud je mimo koridor, přidá se jako nový bod výsledné lomené čáry a spojením s počátečním a koncovým bodem vzniknou dvě nové spojnice, viz obrázek 27. Na obě je rekurzivně aplikován stejný postup. Algoritmus je ukončen ve chvíli, kdy se už žádný jiný bod nenachází mimo koridor, a všechny vzniklé segmenty jsou zpracovány. [14]



Obrázek 27 – Douglas-Peuckerův algoritmus. Převzato z [12]

Algoritmus v krocích:

1. nastavení požadované přesnosti nastavením šířky koridoru h; n = počet bodů linie; $i = 1$,
2. vytvoření imaginárního koridoru ve vzdálenosti h/2 od spojnice bodů P_i a P_n,
3. prohledáváním od bodů P_{i+1} až P_{n-1} se najde bod P_j, který je nejdále od spojnice P_i a P_n,
   3.1. pokud je P_j mimo koridor, vloží se do nové linie; rekurzivně se pokračuje krokem 2 pro $P_i = P_i$ a $P_n = P_j$ a pro $P_i = P_j$ a $P_n = P_n$.
   3.2. Pokud P_j je v koridoru, jsou odstraněny všechny body mezi aktuálními P_i a P_j, $i = j$ a tím je segment.
4. Algoritmus končí po zpracování všech vzniklých úseků linie.

4 Hledání vhodné kombinace algoritmů a struktur

Hledání vhodné kombinace algoritmů a struktur, využitelných ke zpracování surových dat z radaru do formátu popsaného v kapitole 2.3.1, je stěžejní náplní této práce. Hlavním kritériem rozhodujícím o vhodnosti kombinace je doba zpracování dat, a to především kvůli aktuálnosti získaných informací. Nejzazším termínem publikování zpracovaných údajů z dokončené otáčky radaru by měl být časový okamžik dokončení otáčky bezprostředně následující[15]. Tím by bylo zajištěno, že s každou dokončenou otáčkou nebude docházet k nárůstu zpoždění ve zpracování, způsobeného kumulací zpoždění v po sobě jdoucích otáčkách. S narůstajícím zpožděním totiž dochází ke ztrátě hodnoty zpracované informace, neboť v době jejího zveřejnění by již skutečná situace nad sledovaným územím mohla vypadat odlišně.

Jako řešení případného vznikajícího zpoždění by bylo možné podniknout různá opatření, jako například použití výkonnějšího hardware, zvýšení rychlosti zpracování na úkor přesnosti, či vytvořením nejrůznějších strategií, vypořádávajících se s již nastalým problémem, pracujících např. s přeskakováním zpracování informací, které by již v době začátku jejich zpracování nebyly aktuální, apod. Tyto skutečnosti úzce souvisí s konkrétním určením nasazení tohoto zpracování, s povahou systémů, mezi kterými bude pracovat, s požadavky spolupracujících systémů apod. Aspektů, které mají vliv na řešení problémů, vzešlých z nedostatečné rychlosti zpracování, by se dalo najít nepřeberné množství. Tato problematika ovšem není cílem této práce.

Naopak zvýšená pozornost bude věnována různým algoritmům, jejich kombinacím či vlivu datových struktur reprezentujících zpracovávaný obraz, za účelem maximalizace výpočetní efektivity konkrétních případů, s vyhlídkou budoucího možného nasazení jako součást existujících radiolokačních systémů. Integrace do existujících řešení by měla být bez viditelného dopadu na jejich odezvu. Další možností by mohlo být začlenění přímo do hardwaru samotného radaru, tedy do prostředí s omezenou výpočetní kapacitou. To vše ve snaze předejít situacím nastíněným v předchozím odstavci.

Cílem praktické části této práce je ověřit či vyvrátit vhodnost postupu zpracování srážkových informací získaných z primárního radaru, popsaného v kapitole 3, tak, jak byl nastolen jako celek, vhodnost zvoleného filtru pro odstranění šumu ze zpracovávaného obrazu, vhodnost zvoleného algoritmu vyhledávajícího kontury a vhodnost algoritmu zjednodušujícího polygony.

Implementace bude vytvořena taktéž s ohledem na multiplatformní nasazení. Proto byly zvoleny dvě platformy splňující tento požadavek, a to Java a programovací jazyk C++. Pro potřeby porovnání výkonnosti jednotlivých řešení budou tedy vytvořeny dvě alternativní implementace. Pro vývoj v C++ bude použit aplikační framework Qt[16]. V obou zvolených jazycích budou vytvořeny implementace s maximální možnou podporou daného

[15] Trvání otáčky radaru, pro který je aplikace určena, jsou přibližně 4 sekundy.
[16] http://qt-project.org/

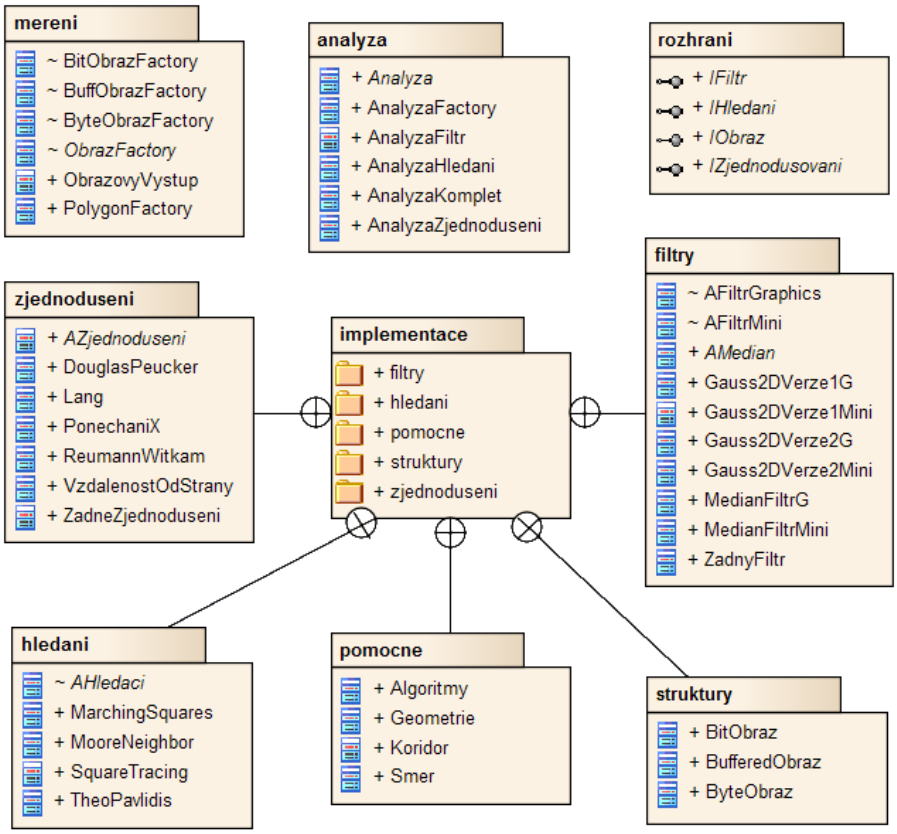
frameworku (především při práci s obrazem) a dále minimalistické implementace datových struktur s ohledem na poskytnutí nejlepšího výkonu pro daný problém.

5 Implementace

V následující kapitole budou představeny zvolené implementace algoritmů a struktur, jejichž vlastnosti budou měřeny. Jako ukázkové budou předvedeny implementace na platformě Java. Případné rozdíly oproti implementaci v jazyce C++ budou uvedeny v závěru kapitoly. K implementaci pomocí Javy bylo použito IDE[17] NetBeans 7.3 a Java SDK 1.7. Implementace v C++ byla realizována ve vývojovém prostředí Qt Creator 2.7 s pomocí SDK Qt 5.0.2.

5.1 Struktura projektu

Testovací projekt je dle zavedených konvencí[18] pro programování na platformě Java rozdělen do několika balíků, obsahujících třídy, týkající se společných problémů. Jedná se o balíky `analyza`, `implementace`, `rozhrani` a `mereni`, přičemž balík implementace je rozdělen do několika dalších balíků, obsahujících třídy s odlišným zaměřením. Strukturu projektu znázorňuje UML diagram balíčků a tříd v nich obsažených nacházející se na obrázku 28.



Obrázek 28 – UML diagram struktury projektu

[17] Integrované vývojové prostředí
[18] http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

5.2 Balík `rozhrani`

Balík `rozhrani` obsahuje pouze rozhraní. Z důvodu maximální možné automatizovatelnosti měření zde byla deklarována čtyři rozhraní, sloužící k definici kontraktu tříd, implementujících měřené algoritmy a datové struktury. Jedná se o následující:

- `IObraz` – rozhraní deklarující společné metody pro datové struktury reprezentující analyzovaný obraz. Jedná se především o metody přistupující k jednotlivým bodům, za účelem zjištění jejich barvy, nastavování barev jednotlivých bodů či metody vracející rozměry obrazu. Všechny implementace tohoto rozhraní se nachází v balíku `implementace.struktury`.

- `IFiltr` – je rozhraní společné pro třídy objektů určených k filtrování obrazu. Všechny třídy implementující vybrané obrazové filtry se nachází v balíku `implementace.filtry`. Jedinou metodou tohoto rozhraní je metoda `filtrujObraz`, přijímající jeden parametr typu `IObraz`, jímž je obraz určený k filtrování, a vracející opět objekt typu `IObraz`, tj. přefiltrovaný obraz.

- `IHledani` – je obdobou rozhraní `IFiltry`, tedy deklaruje metody třídám implementujícím algoritmy pro vyhledávání kontur, nacházejícím se v balíku `implementace.hledani`. Rozhraní je tvořeno také pouze jedinou metodou, a to `najdiKonturu`. Metoda přijímá tři parametry, první typu IObraz a další dva celočíselné. Prvním parametrem je tedy prohledávaný obraz, další dva určují souřadnice bodu v obraze, kterým prohledávání má začít. Návratovou hodnotou je instance třídy `Polygon`.

- `IZjednodusovani` – je posledním rozhraním z balíku `rozhrani` a deklaruje metodu pro třídy implementující algoritmy zjednodušování polygonů, nacházející se v balíku `implementace.zjednoduseni`. Metoda se jmenuje `zjednodus`, přijímá jeden parametr typu `Polygon` a vrací také objekt typu `Polygon`.

5.3 Balík `implementace`

V balíku nazvaném implementace se nachází veškerý funkční kód týkající se měřených algoritmů a struktur. Tento balík jako jediný obsahuje několik dalších balíků. Ty sdružují vždy skupiny tříd implementující stejné rozhraní, jako např. balík `filtry` či `struktury`. Tyto budou rozebrány v následujících podkapitolách.

5.3.1 Filtry

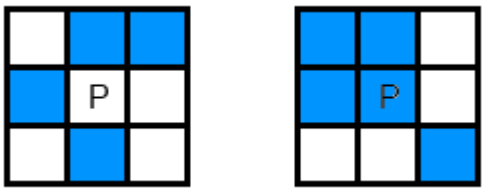
Prvním balíkem v balíku implementace je balík `filtry`. Obsahuje třídy implementující právě dvě různé implementace pro každý z vybraných filtrů. První je implementace celého filtrovacího postupu pro vlastní minimalisticky vytvořené datové struktury, druhá potom

s využitím tříd z Java SDK pro filtrování obrazu reprezentovaného instancí třídy `BufferedImage`.

Filtry k implementaci byly voleny podle jejich teoretických vlastností. Kritériem výběru byla především schopnost filtrů odstraňovat šum v obraze, přičemž nebyl brán ohled na schopnost zachování hran, nebo míru rozmazání, protože cílem není nalézt co nejpřesnější konturu, ale konturu co nejpřesněji vystihující původní tvar objektu v co nejkratším čase a popsat ji pomocí minimálního počtu bodů. Zvolenými filtry jsou medián a dvě masky s Gaussovským rozdělením, a to kvůli jejich jednoduchosti a dobré schopnosti odstranit šum. Gaussův filtr byl zvolen i přes jeho silně rozmazávající účinky, díky jimž se prakticky příliš často nepoužívá. Např. při zpracování digitálních fotografií je silné rozmazání nežádoucí.

Filtr průměr versus filtr medián v dvoubarevném obraze

Při aplikaci filtru, vypočítávajícího průměrnou hodnotu bodu z okolních bodů, na dvoubarevný obraz, je výslednou hodnou reálné číslo z intervalu $\langle 0; 1 \rangle$. Tuto hodnotu je potřeba prahováním upravit, neboť pro dvourozměrný obraz jsou přípustné pouze hodnoty 1 a 0. Pokud by z důvodu jednoduchosti implementace a rychlosti zpracování byl zvolen globální práh 0,5, lze snadno dokázat, že filtr počítající průměr bude mít stejné výsledky jako filtr typu medián. To je způsobeno tím, že ve dvoubarevném obraze je mediánem ta hodnota, která je v uvažovaném obraze častěji a výsledná hodnota průměru uvažovaných hodnot po prahování bude také ta, jichž je mezi průměrovanými hodnotami nejvíce.



Obrázek 29 – Uvažované okolí bodu při aplikaci kovoluční masky o rozměru 3x3 (vlevo obarveny 4 z 9 bodů, vpravo 5 z 9 bodů)

Na Obrázek 29 se nachází vzory krajních případů pro uvažované okolí bodu pro aplikaci konvoluční masky, tedy případy s obarvenými 4/9 a 5/9 bodů. V případě zpracování mediánem obvyklým způsobem, tedy seřazením hodnot uvažovaných bodů, budou výsledky vypadat jako na Obrázek 30. Z obrázku je tedy patrné, že mediánem z okolí bodu ve dvoubarevném obraze je hodnota, která se v okolí bodu vyskytuje častěji.

Obrázek 30 – Medián pro vzory z obrázku 27

V případě průměrovacího filtru v kombinaci s globálním prahem 0,5 je situace stejná. V případě výskytu 4/9 obarvených bodů v uvažovaném prostoru je výsledná hodnota přibližně 0,44. Hodnota je tedy pod prahem a bod bude nastaven jako nevybarvený. V případě výskytu 5/9 obarvených bodů v uvažovaném prostoru je situace obdobná. Výsledná hodnota je přibližně 0,55. Hodnota je tedy nad prahem a bod bude nastaven jako vybarvený.

Z toho důvodu byla zvolena implementace rychlejšího filtru, tedy mediánu. Rychlejší je proto, že při zpracování uvažovaných hodnot je dostatečné spočítat počet výskytů jedné barvy a v případě, že je počet vyšší než polovina počtu bodů v uvažovaném okolí, je možné ukončit zpracování aktuálního bodu a přejít na další.

5.3.2 Hledani

Balík `hledani` obsahuje několik různých implementací rozhraní `IHledani`. Jedná se o algoritmy, vyhledávající v dvoubarevném obraze kontury objektů. S jednou výjimkou byly implementovány algoritmy představené v kapitole 3.3. Zmíněnou výjimkou je algoritmus nazvaný Radial Sweep, neboť i přes odlišný přístup k prohledávání okolí bodu prakticky téměř totožný s algoritmem Moore-Neighbor Tracing. Z těchto dvou byl implementován druhý zmíněný, a to z důvodu lepší volby bodu, kterým začíná prohledávání okolí. Náročnost je tak v každém případě o jeden přístup k bodům obrazu nižší. Kromě tříd obsahujících konkrétní implementace, obsahuje balík také jedu abstraktní třídu, `AHledani`, která je v hierarchii dědičnosti společným předkem všech vyhledávacích algoritmů a obsahuje funkcionalitu sdílenou všemi čtyřmi algoritmy. Příkladem takové společné funkcionality je hledání počátečního bodu kontury, jako výchozího pro samotné algoritmy. Na rozdíl od filtrů jsou všechny vyhledávací algoritmy implementovány pouze jednou, neboť veškeré rozdíly v použitých datových strukturách byly abstrahovány použitím rozhraní `IObraz`.

5.3.3 Zjednoduseni

V balíku `zjednoduseni` jsou soustředěny vybrané implementace zjednodušovacích algoritmů. Realizovány jsou všechny algoritmy představené v kapitole 3.4 a z důvodu zjednodušení měření také obsahuje jednu implementaci, která polygon nijak nezjednoduší, ale vrátí původní tvar. Implementace zjednodušovacích algoritmů byla provedena pouze jednou, neboť není nijak vázána na struktury použité v předchozích krocích analýzy, jejichž výstupem jsou již data ve stejném formátu, tj. instance třídy `Polygon`.

5.3.4 Struktury

Balík struktury obsahuje tři implementace rozhraní `IObraz`, tedy tři různé reprezentace obrazu, z nichž jedna je pouze zapouzdřením pro reprezentaci obrazu třídou z frameworku Javy. Jedná se o následující třídy:

1. `BufferedObraz` – třída zapouzdřující třídu `BufferedImage` z Java frameworku, z balíku `java.awt.image`. Třída `BufferedImage` je postavená na třídách `ColorModel` a `Raster`, tedy reprezentaci barevného

modelu a rastru bodů. Použitý barevný model je typu realizovaného konstantou `TYPE_BYTE_BINARY` třídy `BufferedImage`, který použitím bez specifikace barevného modelu vede k vytvoření jednobitového[19], tedy dvoubarevného obrazu.

2. `ByteObraz` – je první vlastní implementací obrazových dat, navržená k minimalizaci času potřebného k přístupu na konkrétní bod. Nejmenší využitou jednotkou je byte, reprezentující jeden bod obrazu a nesoucí informaci o jeho barvě. Struktura je koncipována jako dvoubarevný obraz, jednotlivé byty tedy nesou hodnotu 0 nebo 1. Celý rastr je pak reprezentován dvourozměrným polem bytů, aby mohl být přístup k jednotlivým bodům obrazu realizován přímou adresací pole pomocí indexů, které tak odpovídají souřadnicím *x* a *y* daného bodu.

3. `BitObraz` – je druhou vlastní implementací jednobitového obrazu, tentokráte navržená k minimalizaci paměťové náročnosti struktury, za cenu mírně zvýšené náročnosti přístupu k hodnotě konkrétního bodu. Implementace je založena taktéž na dvourozměrném poli bytů, avšak se snahou o maximalizaci využitelnosti jednotlivých bytů. Toho je dosaženo tím, že téměř každý[20] byte obsahuje informaci o hodnotě osmi sousedících bodů. Adresace bodů je řešena tak, že jeden rozměr, výška obrazu, je adresována přímo indexem pole, takže jeden rozměr pole odpovídá rozměru obrazu. Druhý přístup je mírně zkomplikován využitím všech bitů tak, že je nutno nejdříve, celočíselným dělením pořadí požadovaného bodu osmi, vypočíst index bytu, obsahujícího hledanou informaci. Indexem bitu v bytu je pak zbytek po zmíněném celočíselném dělení.

5.3.5 Pomocne

Je balíkem obsahujícím různorodě zaměřené třídy, nehodící se svým zařazením mezi žádné jiné konkrétní třídy v jiném balíku. Zástupcem těchto tříd jsou např. třída `Geometrie`, obsahující např. metodu pro výpočet vzdálenosti bodu od přímky či výčtová třída `Smer`, zavádějící konstanty reprezentující směr orientace, např. pro vstup na bod, či metody pracující se směrem, jako např. změna směru o 90° vlevo. Tyto třídy poskytují své členské konstanty a metody třídám napříč celým projektem.

5.4 Balík `analyza`

Balík `analyza` obsahuje třídy, které jsou všechny svým způsobem spojené s analýzou obrazu. Je místem propojujícím ostatní části projektu do větších spolupracujících celků. Dílčí algoritmy skládá za sebe, čímž tvoří konkrétní části analýzy. Měření samotné proběhne ve dvou fázích. Těmito fázemi budou měření každého algoritmu zvlášť a měření všech možných kombinací algoritmů, tvořících celý proces analýzy. Balík se skládá z jedné abstraktní třídy, nazvané `Analyza`, z jedné tovární třídy, pojmenované

[19] Více informací lze najít v dokumentaci třídy BufferedImage, např. na adrese http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/image/BufferedImage.html.
[20] S výjimkou bytů na okrajích jednoho rozměru, které v případě nedělitelnosti rozměru osmi, obsahují několik nevyužitých bitů.

`AnalyzaFactory`, sloužící k vytváření instancí různých analýz a kombinací jejich dílčích kroků, a ze čtyř tříd reprezentujících tři dílčí části analýzy a kompletní analýzu, jako zapouzdření dílčích.

Třída `Analyza`, jako předek všech tříd představujících různé části analýzy, definuje několik společných metod, určených k usnadnění měření rychlosti zpracování jednotlivými algoritmy a jednu virtuální metodu, kterou jsou potomci povinni implementovat. Touto virtuální metodou je bezparametrická metoda analyzuj, vracející hodnotu datového typu long, vyjadřující čas provádění analýzy.

5.5 Balík `mereni`

V balíku `mereni` jsou koncentrovány třídy podílející se na průběhu měření a generování výstupů. Na tomto místě se potkávají třídy ze zbytku projektu a dohromady vytváří proces, při kterém dochází ke kombinování algoritmů, analýze vstupů a ukládání naměřených výsledků a obrazových výstupů.

Balík obsahuje několik továrních tříd, které načítají ze souborů a připravují vstupní data pro měřené algoritmy, třídu zajišťující vygenerování výstupu algoritmů do png souborů[21], ve kterých jdou na jednom místě vedle sebe vidět výstupy jednotlivých kroků tvořících aktuální měřenou kombinaci algoritmů a třídu `Main`, obsahující metodou `main`, která je vstupním bodem programu.

5.6 Rozdíly v implementacích Java a C++

Rozdílů mezi oběma implementacemi je obrovské množství, vycházející z odlišné povahy obou zvolených platforem. Obě alternativy se prakticky shodují pouze z algoritmického hlediska a z pohledu implementovaných skutečností, stejně tak jako v návrhu a struktuře jednotlivých programovaných tříd. V následujících odstavcích budou tedy představeny pouze nejdůležitější rozdíly.

5.6.1 Třídy reprezentující rastrový obraz

K tvorbě obou projektů byly z použitých frameworků využity k několika účelům třídy reprezentující rastrový obraz. Konkrétně jde o třídu `BufferedImage` na platformě Java a o třídu `QImage` z frameworku Qt. Obě jsou si svou podstatou velmi podobné, tedy obě využívají jako podkladovou paměťovou strukturu jednorozměrné pole bytů[22], obě dokáží reprezentovat obraz různé barevné hloubky apod. V této práci byly využity především k následujícím účelům.

- Načítání a ukládání obrazů ve formátu png, které by bez jejich pomoci bylo obtížné.

[21] Kompletní sada výstupů pro kombinace algoritmů se nachází na přiloženém DVD.
[22] C++ neobsahuje přímo datový typ byte, ale několik typů o rozsahu jednoho bytu. V tomto případě je využit datový typ unsigned char. Více informací v dokumentaci zde http://qt-project.org/doc/qt-4.8/qimage.html.

- Jako zapouzdřená struktura k reprezentaci meteo obrazu, viz kapitola 5.3.4.

S použitím těchto tříd úzce souvisí použití tříd umožňujících manipulaci s obrazem, jako např. vykreslení bodu, vykreslení jiného obrazu apod. Jsou využity především ke kreslení do obrazu. Jedná se o třídy `Graphics` (Java) a `QPainter` (Qt).

5.6.2 Dealokace paměti

Dalším velkým rozdílem je způsob správy operační paměti. V projektu vytvořeném v jazyce C++ musí být na rozdíl od platformy Java brán zřetel na její důslednou dealokaci. Toto je zapříčiněno tím, že platforma Java využívá automatickou správu paměti, kterou zajišťuje část JVM, zvaná garbage collector. Jedná se o mechanismus sledování paměti alokované na haldě[23], který v okamžiku, kdy zjistí, že na některý objekt již neexistuje žádná reference, tzv. mrtvý objekt vymaže a paměť jím zabranou označí jako prázdnou. Tento úklid paměti probíhá v rámci JVM automaticky, ale může být také explicitně vyvolán např. zavoláním, `System.gc()`. V měřícím projektu je využita i druhá možnost, a to po každé dokončené replikaci měření. Ve standardním C++ žádný podobný mechanismus neexistuje, a proto je potřeba již nepotřebné objekty mazat explicitně, např. pomocí unárního operátoru `delete`, jehož jediným operandem je ukazatel na objekt. Jeho použití způsobí zavolání destruktoru objektu, na který ukazoval daný ukazatel, který je také potřeba naprogramovat. V případě ignorování dealokace paměti v jazyce C++ by velice rychle došlo k přetečení paměti a následnému pádu programu.

5.6.3 Měření doby zpracování

Doba zpracování je v obou projektech měřena v nanosekundách, a to především kvůli rychlosti některých měřených operací, které jsou někdy rychlejší než 1ms, a tedy výsledkem jejich změření by bylo 0ms. Zjištění uplynulého času v nanosekundách je na obou platformách implementován mírně odlišně.

Na platformě Java je doba trvání měřené operace určena odečtením systémového času před začátkem operace od systémového času po skončení operace. Systémový čas vyjádřený v nanosekundách lze získat pomocí zavolání `System.nanoTime()`. Oproti tomu framework Qt obsahuje třídu přímo určenou k měření uplynulého času mezi dvěma okamžiky, nazvanou `QElapsedTimer`. Její použití je snadné a spočívá v zavolání metody `start()` pro určení počátku měření a v zavolání metody `nsecsElapsed()` na konci měření. Návratovou hodnotou druhé zmíněné metody je pak 64bitové číslo vyjadřující trvání mezi voláním obou metod v nanosekundách.

[23] Více informací o způsobu alokace paměti v JVM v dokumentaci platformy Java zde: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html.

6 Výstupy

V této kapitole budou na ukázkových obrazech představeny výstupy jednotlivých implementovaných filtrů, vyhledávacích a zjednodušujících algoritmů, a to bez ohledu na podkladovou strukturu[24]. Výstupy různých kombinací těchto algoritmů nebudou z důvodu nedostatku místa uvedeny. Kompletní sada výstupů, včetně kombinací struktur, se nachází na přiloženém DVD. Ukázka výstupů napříč vybranou kombinací algoritmů je v příloze A této práce.

6.1 Výstupy filtrů obrazu

Na následujících třech obrázcích jsou vidět efekty tří implementovaných filtrů. Požadavek na odstranění šumu a složitosti obrysů splňují dle očekávání všechny zvolené filtry, přičemž nejlepšího výsledku dosáhl filtr využívající mediánu.

Obrázek 31 – Efekt filtru s Gaussovským rozdělenými váhovými koeficienty dle vzorce 3.5, vlevo

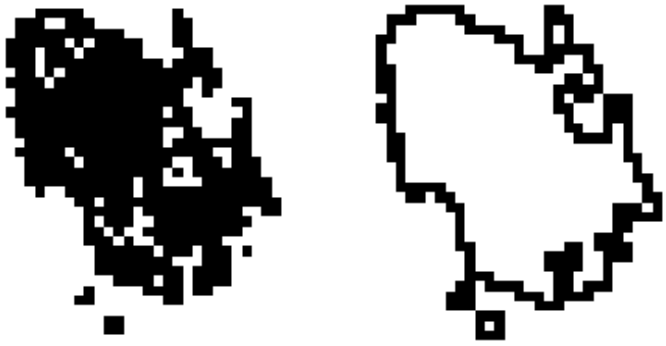
Obrázek 32 – Efekt filtru s Gaussovským rozdělenými váhovými koeficienty dle vzorce 3.5, vpravo

[24] Použitá datová struktura nemá vliv na výsledek práce algoritmů.

Obrázek 33 – Efekt mediánového filtru

6.2 Výstupy vyhledávacích algoritmů

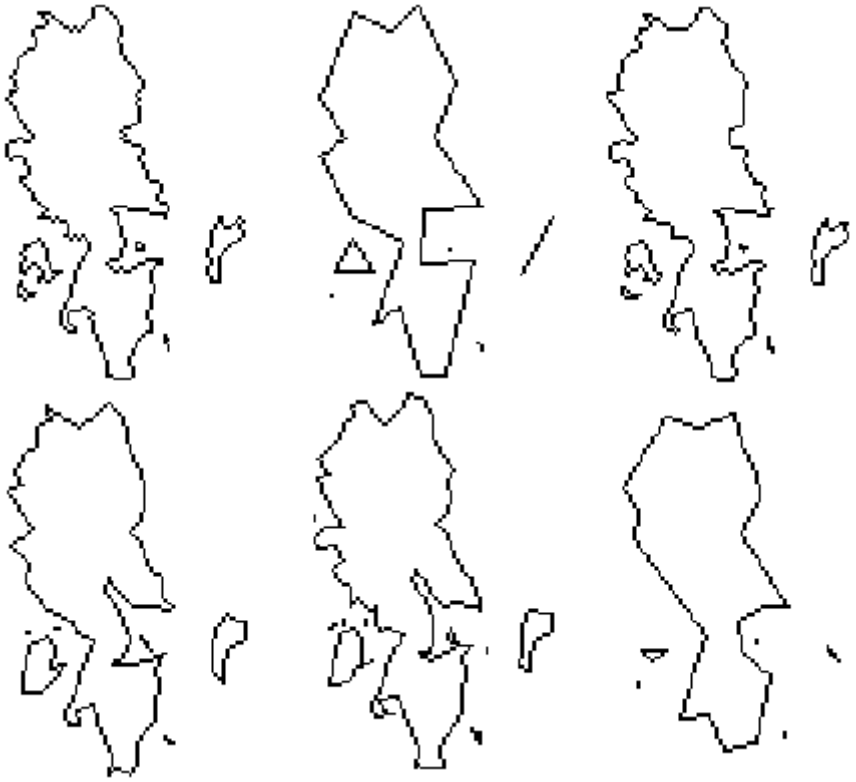
Výstupy vyhledávacích algoritmů jsou všechny stejné, což vyplývá z jejich podstaty. Cílem všech je totiž nalézt všechny objekty v obraze. Jediným rozdílem je pouze počet objektů, do kterých daný algoritmus rozdělí zpracovaný obraz a samozřejmě postup, kterým je nalezení objektů dosaženo.



Obrázek 34 – Příklad výstupu vyhledávacích algoritmů

6.3 Výstupy zjednodušujících algoritmů

Algoritmů patřících do této skupiny je nejvíce a z důvodu jejich parametrizovatelnosti téměř neomezeným množstvím parametrů, budou uvedeny výstupy pouze pro jednu hodnotu. Jelikož parametry všech těchto algoritmů vyjadřují svým způsobem vzdálenost mezi obrazovými body, jsou všechny uvedené výstupy vybrané pro stejnou hodnotu parametru.



Obrázek 35 – Výstupy zjednodušovacích algoritmů

Obrázek 35 znázorňuje vlevo nahoře originální nezjednodušenou konturu. Dále zleva doprava, shora dolů výstupy algoritmů Douglas-Peucker, Vzdálenost od úsečky, Ponechání bodu, Lang a Reumann-Witkam.

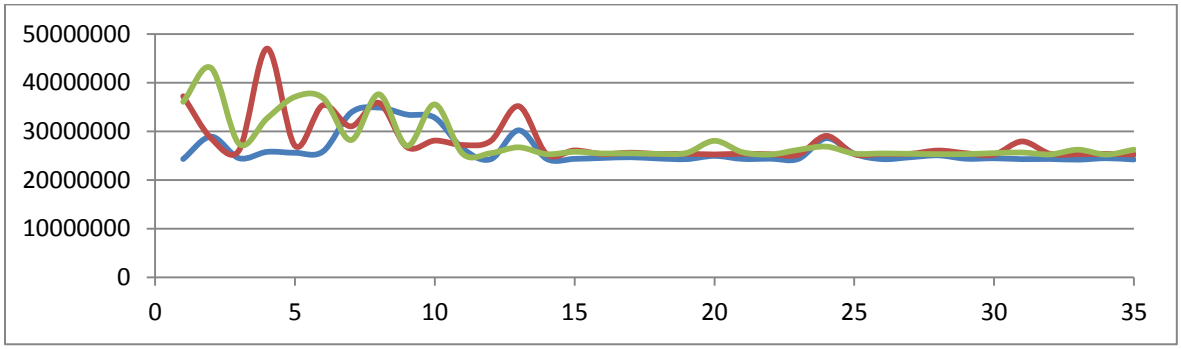
7 Prezentace naměřených výsledků, zhodnocení

Poslední kapitola této práce představí postup měření výkonnosti jednotlivých kombinací algoritmů a datových struktur a budou zde prezentovány naměřené výsledky. Nejdříve zvlášť pro každou skupinu algoritmů a poté pro kombinace tvořící celý postup zpracování.

7.1 Postup měření

Pro účely této práce byl vytvořen jednoduchý měřící algoritmus, jehož podstatou je opakované vytváření kombinací algoritmů a struktur, příprava vstupních dat a spouštění zpracování těchto dat dříve vytvořenou kombinací. Čas tohoto zpracování je dočasně uložen v operační paměti a na konci každé replikace je vytvořen výstupní soubor, do kterého jsou naměřené výsledky zaznamenány. Jednu replikaci tvoří změření času zpracování všech vytvořených kombinací a změření času zpracování pro jednotlivé algoritmy samostatně, nezávisle na ostatních. Měření probíhá v nanosekundách, viz kapitola 5.6.3.

Zvolený počet replikací měření byl 100. Z grafu průběhu ustalování průměru časů, naměřených pro tři náhodně vybrané kombinace algoritmů, zobrazeného na obrázku 36, vyplývá, že dostatečným počtem replikací, po kterých dojde k ustálení průměrné hodnoty naměřeného času, je přibližně 30 replikací.



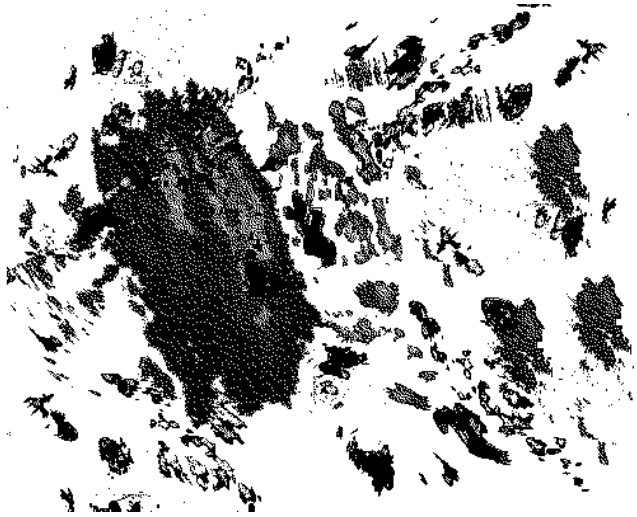
Obrázek 36 – Ustalování průměrného naměřeného času vybrané kombinace algoritmů v průběhu sta replikací

7.2 Vstupní data

Vstupní data byla připravena na základě reálných radarových dat a koncentrována do dvou souborů. Byla vytvořena s ohledem na principy fungování především vyhledávajících a zjednodušujících algoritmů, tak, aby se vstupní soubor svou složitostí ke zpracování blížil extrémním případům, jejichž zpracování by mělo teoreticky být časově nejnáročnější. Záměrně nebylo zmíněno filtrování, neboť na jeho rychlost jeho zpracování nemá obsah obrazu vliv.

Z podstaty algoritmů vyhledávajících kontury objektů v obraze (viz kapitola 3.3) je patrné, že základní skutečností, ze které lze odvodit časovou náročnost vyhledání objektů v předloženém obraze, je součet obvodů kontur všech objektů, které se v obraze nachází.

Na zmíněném součtu obvodů obrysů závisí i zjednodušovací algoritmy (viz kapitola 3.4). Z toho lze vyvodit, že časově nejnáročnějšími obrazy ke zpracování celým procesem budou obrazy, obsahující velké množství objektů se složitými obrysy, složenými z velkého množství bodů. Jako zástupce takového obrazu byl připraven vstup znázorněný na obrázku 37. Dále byl jako referenční vytvořen vstup opačného charakteru, tedy obraz s velkým podílem volné plochy a malým množstvím objektů, viz obrázek 38.



Obrázek 37 – Obrazový vstup ke zpracování za účelem měření časové náročnosti vybraných algoritmů

Obrázek 38 – Referenční obrazový vstup

7.3 Testovací stroje

Měření samotné bylo provedeno na dvou strojích. Prvním byl běžně dostupný počítač osazený dvou jádrovým procesorem Intel Core i3 s frekvencí 2,4GHz, 8GB operační paměti a operačním systémem Windows 7 Home Premium. Pro účely ověření možnosti nasazení vytvořeného procesu zpracování radarových meteoinformací v prostředí s omezenou výpočetní kapacitou, bylo měření provedeno také na druhém, méně výkonném

počítači. Tím se stal přibližně 15 let starý osobní počítač, disponující procesorem Intel Celeron o frekvenci 500MHz osazený 256MB operační paměti. Operačním systémem byl Microsoft Windows XP Professional.

Příkladem prostředí s omezeným výkonem, ve kterém by proces zkoumaný v této práci mohl být nasazen, by mohl být např. samotný radarový systém, který meteoinformace získal. Zpracování by v něm mělo běžet bez dopadu na jeho odezvu.

7.4 Paměťová náročnost datových struktur

Přestože nedostatek operační paměti u osobních počítačů není v dnešní době nikterak závažným problémem a objem dat reprezentujících dvoubarevný rastr není příliš velký, nebylo by na místě paměťovou náročnost zcela ignorovat. Zcela jiná situace s operační pamětí by nastala v případě implementace procesu např. do programovatelného hradlového pole, neboli FPGA[25]. Kapacity pamětí takových obvodů pak nepřevyšují jednotky MB.

7.4.1 Datová struktura BitObraz

Základem datové struktury nazvané `BitObraz` je dvourozměrné pole bytů a jednotkou nesoucí informace o jednom bodě je jeden bit. Pole je uspořádáno tak, aby zabíralo minimální paměťový prostor, při zachování rychlosti přístupu k jednotlivým bodům. První rozměr pole je shodný s výškou obrazu a je adresován přímo, zatímco druhý rozměr pole je určen vydělením šířky obrazu osmi a následným zaokrouhlením na nejbližší celé číslo nahoru. Například pro obraz o rozměrech 10x15 bodů, bude alokováno 20 bytů, tedy pole 10x2 byty. V ideálním případě, tedy když bude šířka obrazu dělitelná beze zbytku osmi, bude využit každý alokovaný bit. V nejhorším případě zůstane pro každý řádek obrazu nevyužito sedm bitů, a to pokud zbytek po vydělení šířky osmi bude roven jedné.

7.4.2 Datová struktura `ByteObraz`

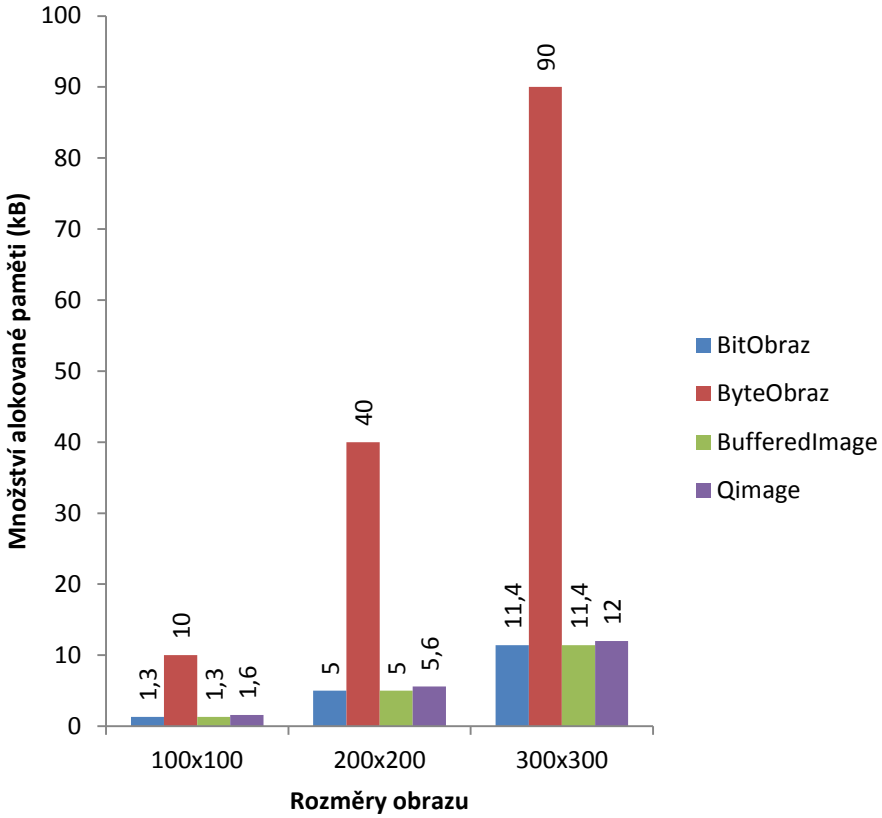
Tato datová struktura je taktéž založena na dvourozměrném poli bytů, avšak s tím rozdílem, že jeden byte obsahuje informaci pouze o jednom bodě obrazu. Alokováno tedy bude tolik bytů, kolik má obraz bodů. Tato struktura je z hlediska alokované paměti neefektivní, neboť je reálně využita pouze jedna osmina alokované paměti. Struktura byla do porovnání zahrnuta z toho důvodu, že jsou jednotlivé byty adresovány v obou rozměrech přímo, bez nutnosti přepočtu mezi souřadnicemi bodů obrazu a pole bytů.

7.4.3 `BufferedImage` a `QImage`

Obě dvě struktury, poskytnuté odpovídajícími frameworky, používají k uložení informace o barvě jednotlivého bodu pouze jeden bit a jako podkladová datová struktura jim oběma slouží jednorozměrné pole bytů. I přes tyto společné vlastnosti existuje rozdíl ve způsobu alokace paměti. Zatímco `BufferedImage` alokuje paměť shodným způsobem jako struktura `BitObraz`, u třídy `QImage` je ve většině případů alokováno paměti více. Je to způsobeno zarovnáváním alokované paměti pro šířku obrazu do bloků po 32 bitech. V nejlepším případě, kdy bude šířka obrazu dělitelná beze zbytku 32, bude využit každý

[25] Více na http://cs.wikipedia.org/wiki/Programovateln%C3%A9_hradlov%C3%A9_pole.

alokovaný bit. V nejhorším případě, tedy když bude zbytek po celočíselném dělení šířky obrazu 32 roven jedné, zůstane v pro každý řádek obrazu nevyužito 31 bitů. [16] [17]



Obrázek 39 – Graf množství alokované paměti jednotlivými strukturami pro obrazy různých rozměrů

Na obrázku 39 je vidět množství alokované paměti jednotlivými strukturami pro čtvercové obrazy o délkách stran 100, 200 a 300 bodů. Nejúspornějšími strukturami jsou dle předpokladů struktury `BitObraz` a `BufferedImage`, následované strukturou `QImage`. Struktura `ByteObraz` je dle očekávání z hlediska alokované paměti velmi neefektivní.

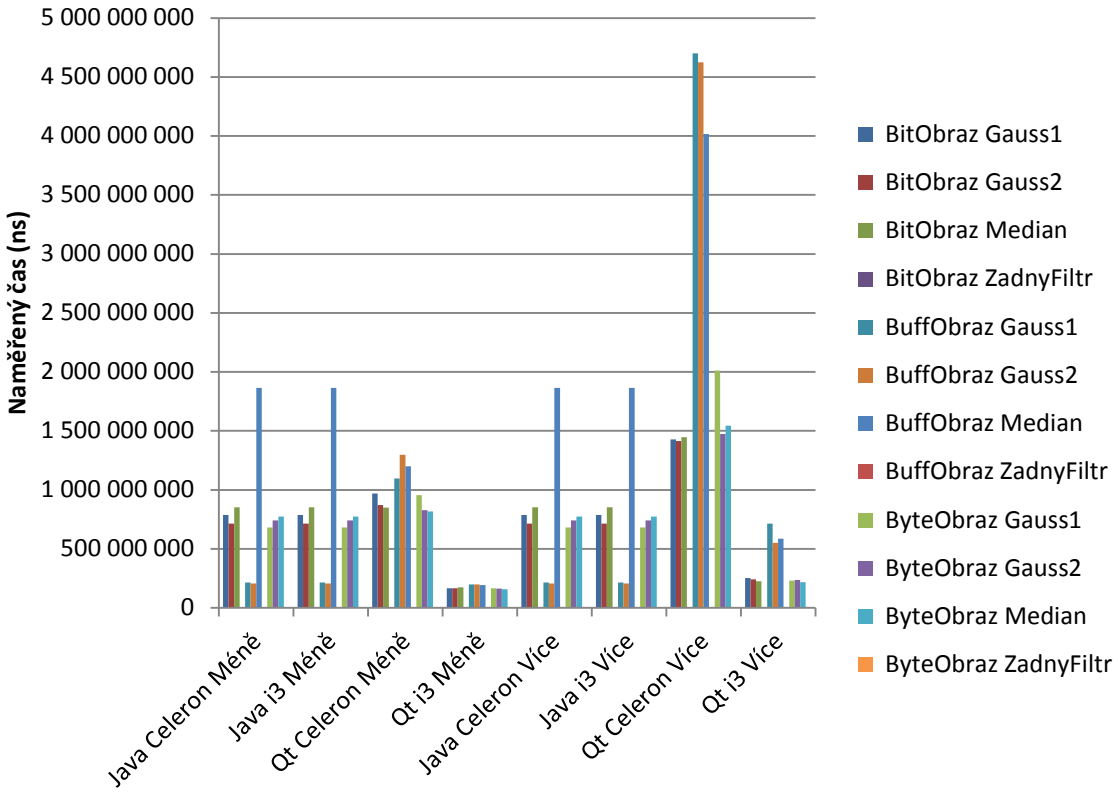
7.5 Rychlosti skupin algoritmů

Následující podkapitoly práce budou věnovány prezentaci průměrných naměřených rychlostí samostatně stojících algoritmů, v rámci jednotlivých skupin. Rozdíly v měřeních dvou testovacích obrazů jsou u jednotlivých platforem vyznačeny hesly „Více“ u složitějšího obrazu a „Méně“ u jednoduššího obrazu. Kompletní soubory naměřených časů pro všechny replikace se nachází na přiloženém DVD.

7.6 Rychlost filtrů

Jak jde vidět z grafu průměrných rychlostí na obrázku 40, jsou rychlosti zpracování filtrů ve většině zvažovaných případů velice podobné. Jako nejrychlejší se na první jeví implementace filtrů na platformě Qt. Je tomu tak ale pouze v případě měření na moderním

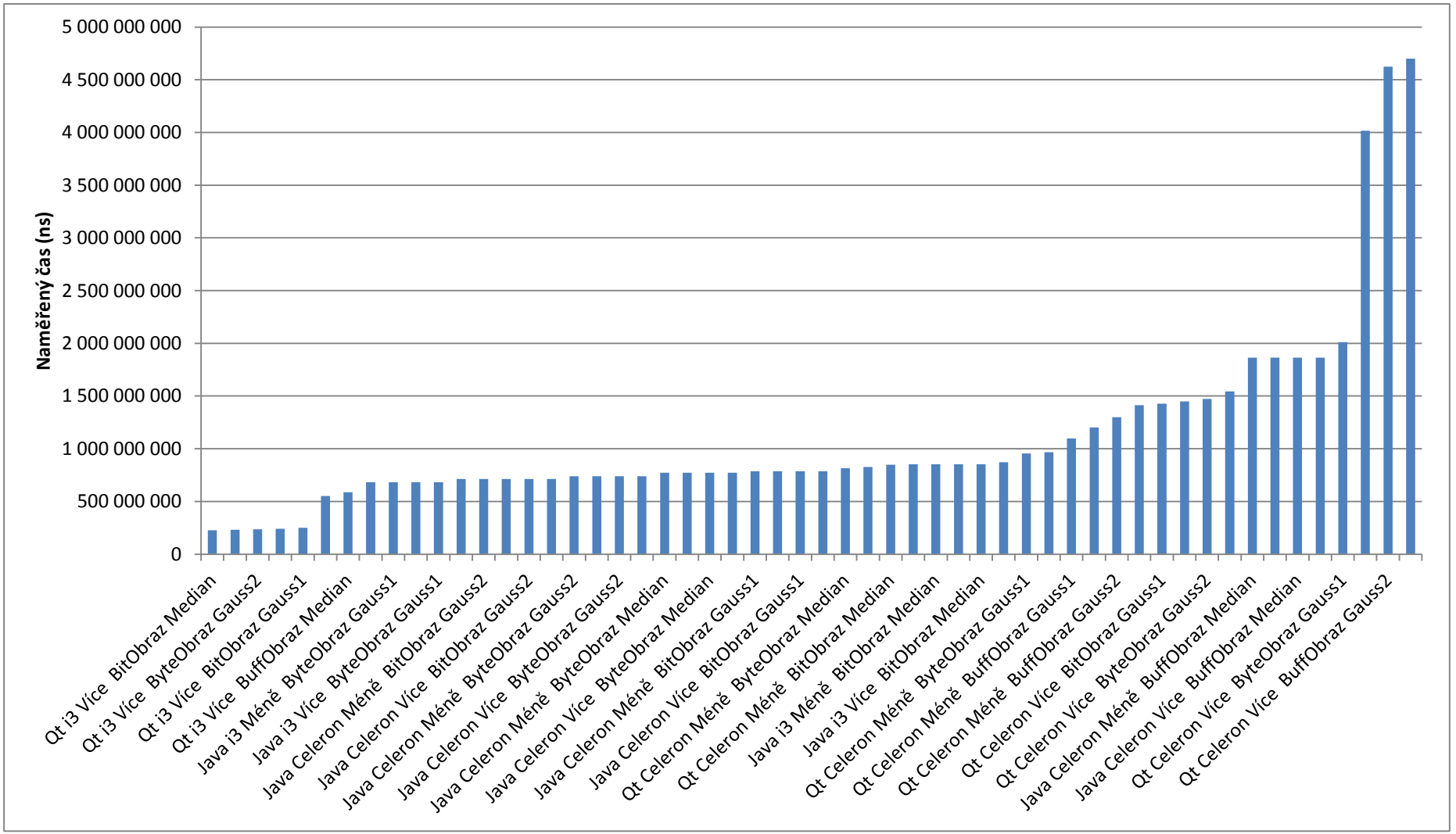
procesoru Intel Core i3. Na pomalejší platformě s procesorem Intel Celeron je výrazně pomalejší.



Obrázek 40 – Graf rychlostí jednotlivých fitrů na různých strukturách

Naopak velice konzistentní výsledky, rychlostí srovnatelné s nejrychlejším zpracováním v QT, poskytuje implementace konvolučních filtrů pomocí zabudovaných tříd z SDK platformy Java. Čas potřebný pro jejich aplikaci se na obou měřených strojích pohybuje průměrně okolo 200ms. Nepoužití filtru je napříč všemi implementaci realizováno za dobu kratší než 10µs.

Na obrázku 41 je vidět graf, ve kterém jsou naměřené hodnoty seřazené dle naměřené rychlosti od nejmenší po největší. Do tohoto grafu nejsou zavedeny hodnoty naměřené bez aplikace filtru.

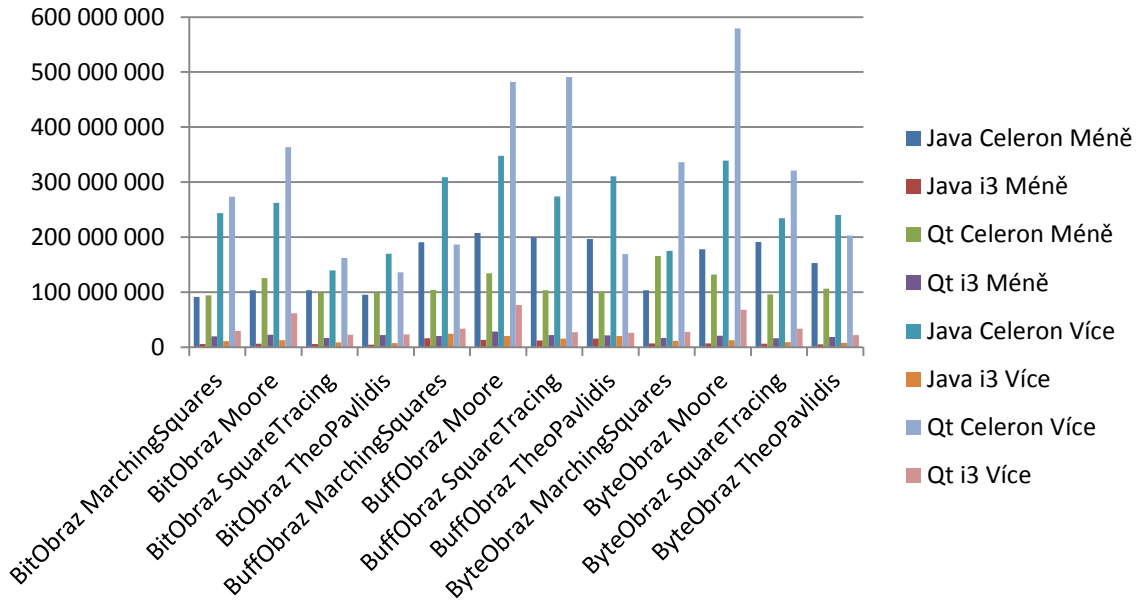


Obrázek 41 – Graf rychlostí filtrů seřazený dle naměřených rychlostí

7.7 Rychlost vyhledávacích algoritmů

Algoritmy vyhledávající kontury objektů, určené k měření výkonnosti, byly vybrány čtyři, tj. všechny algoritmy popsané v kapitole 3.3. Tato část zpracování je nejdůležitější, neboť se fakticky jedná o jádro celého problému.

Na obrázku 42 se nachází graf naměřených rychlostí vyhledávacích algoritmů, seskupený do osmi skupin, podle toho na jakém obraze, procesoru a platformě byly naměřeny. Je z něj patrné, že největší vliv na rychlost zpracování má použitý procesor. Doba zpracování pomalejším procesorem je v závislosti na zpracovávaném obraze a použitém algoritmu přibližně 10-60x vyšší, než v případě použití moderního procesoru.

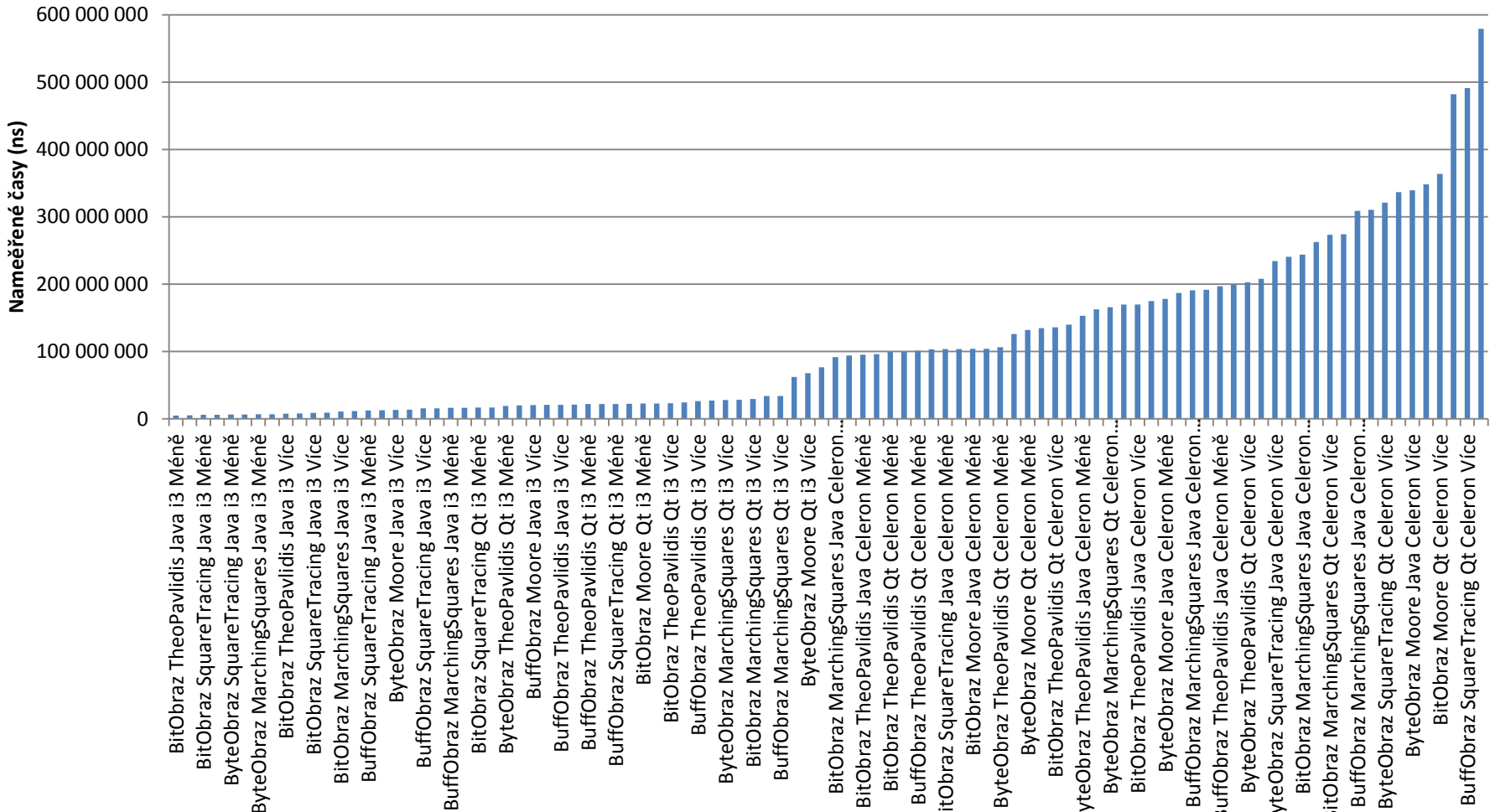


Obrázek 42 – Graf naměřených časů ruzných kombinací algoritmů, seskupený dle procesoru, platformy a zpracovaného obrazu

Při použití rychlejšího procesoru jdou vidět pouze nepatrné odchylky mezi zpracovávaným obrazem a použitým algoritmem.

Graf na obrázku 43 jasně dokazuje několikanásobně vyšší rychlosti zpracování pomocí moderního procesoru. Použitý procesor prakticky rozděluje graf na dvě poloviny, kdy moderní procesor Core i3 obsazuje celou rychlejší polovinu grafu.

Při odhlédnutí od použitého procesoru vychází jako nejrychlejší, bez ohledu na použitou datovou strukturu a platformu, algoritmus Theo Pavlidis. Velice dobré výsledky a prakticky druhé nejrychlejší časy poskytuje nejjednodušší z implementovaných algoritmů, tedy Square Tracing.



Obrázek 43 – Graf naměřených hodnot seřazený od nejrychlejšího

Z pohledu použitých datových struktur se jako nejrychlejší jeví vlastní implementace, nazvaná **BitObraz**. I přes vysokou míru shodnosti s implementacemi obrazu z použitých SDK, jsou tyto třídy znatelně pomalejší. To je způsobeno tím, že přistup k hodnotě pixelu struktur z SDK je ovlivněn jejich univerzálností a možností nasazení pro různé barevné hloubky obrazu. Prakticky to znamená, že se struktura při dotázání na hodnotu pixelu musí nejdříve dle použitého formátu obrazu rozhodnout, kde je požadovaná hodnota uložena. Pro urychlení této operace existuje na obou platformách možnost přistupovat k obrazové struktuře přímo, jako k poli hodnot. Použitím tohoto přístupu by se struktury staly prakticky shodnými s implementací **BitObraz**. Tento postup je však z programátorského hlediska výrazně složitější, neboť programátor přebírá veškerou zodpovědnost za přístup k jednotlivým hodnotám pixelů, za rozhodnutí o formátu obrazu, dle kterého budou hodnoty interpretovány atd.

7.8 Rychlost zjednodušujících algoritmů

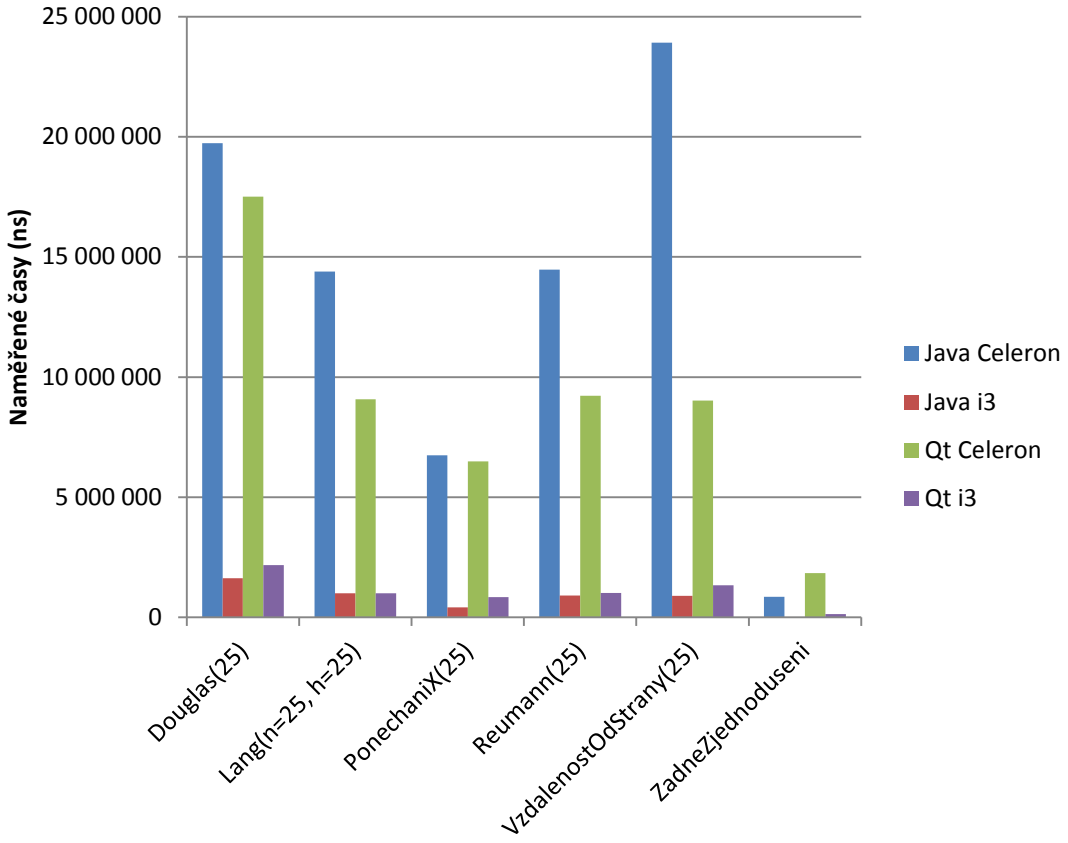
Objektivní srovnání rychlostí této skupiny algoritmů je díky jejich parametrizovatelnosti a odlišným výstupům pro stejné hodnoty parametrů (viz kapitola 6.3) prakticky nemožné. Především záleží na subjektivním názoru na to, jaký výstup je ještě dostatečně přesný a s jakými parametry daného algoritmu jich bylo dosaženo. Teprve ve chvíli, kdy by byly pro každý algoritmus nalezeny hodnoty parametrů, dávající srovnatelné výsledky, odpovídající požadovanému výstupu, bylo by měření objektivní. Z důvodu značné náročnosti tohoto hledání a teoreticky neomezeného množství možných kandidátských parametrů bylo měření těchto algoritmů zjednodušeno. Byl zvolen parametr o hodnotě 25 pixelů, který u všech algoritmů vyjadřuje přibližnou vzdálenost mezi body k porovnání, čili slouží jako kritérium, určující zda je daný bod ještě významný nebo už ne. 25 pixelů je přibližně jedna dvacetina rozměrů zpracovaných obrazů.

Způsob srovnání pro účely této práce, kde je pro celý proces zpracování dat nejdůležitější krok vyhledání kontur objektů, je značně orientační, neboť důkladné porovnání výkonnosti těchto algoritmů by se mohlo stát náplní jiné práce.

Jako vstup pro měření rychlosti zjednodušovacích algoritmů byl připraven textový soubor[26], obsahující na každém řádku souřadnice bodů jednoho polygonu. Tento soubor obsahuje celkem 438 polygonů a k jeho vytvoření byl použit implementovaný algoritmus Moore-Neighbor aplikovaný na obrázek 37, přičemž jeho výstupy byly zapsány do souboru.

Z grafu na obrázku 44 je patrné, že pro zvolený parametr algoritmů jsou výsledky jednoznačné z hlediska použitého procesoru. Slabšímu procesoru trvalo zpracování přibližně 5-10x déle, dle uvažovaného algoritmu. V této skupině algoritmů je také patrný nezanedbatelný vliv zvolené platformy, viditelný především na rozdílech v rychlosti algoritmů na slabším procesoru. Jako výkonnější se jeví použití jazyka C++, které v extrémním případě podává na slabším procesoru až 2x rychlejší výkon, než řešení v Javě.

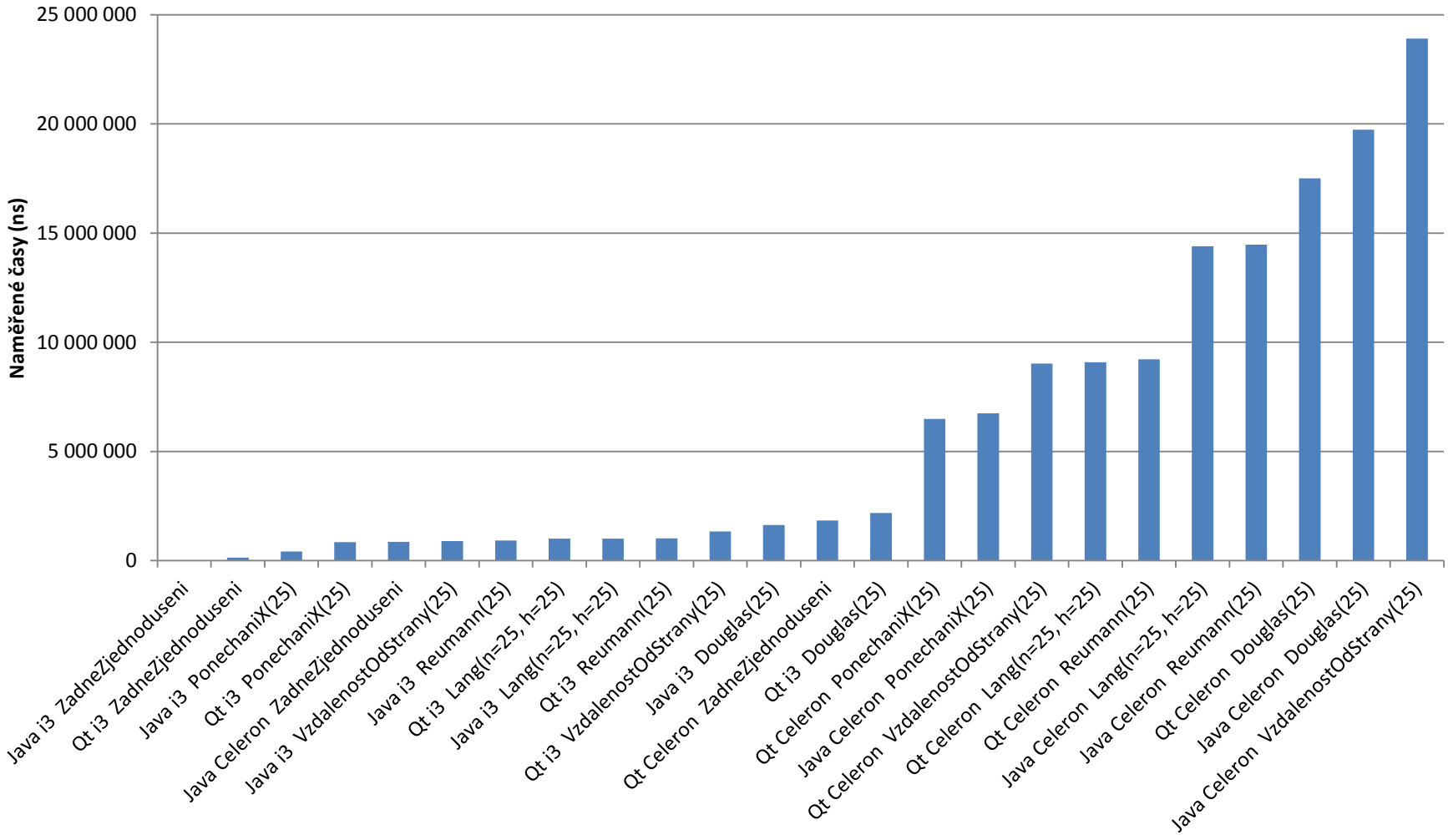
[26] Soubor se jmenuje polygony.txt a je k dispozici na přiloženém DVD.



Obrázek 44 – Skupinový graf rychlostí zjednodušovacích algoritmů

Datová struktura použitá pro celý proces nemá na rychlost zjednodušování vliv, neboť tento krok již není prováděn nad obrazem, ale nad sekvencemi bodů, vyjádřených celočíselnými souřadnicemi. Proto na ní není brán zřetel.

Kompletní výsledky, seřazené dle naměřených, časů jsou zobrazeny v grafu na obrázku 45. Nejrychlejším algoritmem je podle očekávání nejjednodušší implementovaný algoritmus, ponechávající ve výsledné kontuře každý *n*-tý bod, těsně následovaný algoritmem počítajícím kolmou vzdálenost bodu, od spojnice jeho sousedů. Z algoritmů používajících lepší heuristiky a zohledňujících tvar celé linie nelze určit jednoznačně ten nejrychlejší. Prakticky totožné časy byly naměřeny u Langova a Reumann-Witkamova algoritmu. Nejpomalejším algoritmem napříč platformami i procesory byl algoritmus Douglas-Peucker. Za povšimnutí také stojí, že na moderním rychlém procesoru bylo zpracování některých algoritmů rychlejší, než pouhé vrácení celé linie bez zjednodušení na procesoru pomalejším.



Obrázek 45 – Graf seřazených naměřených rychlostí zjednodušovacích algoritmů

7.9 Rychlost kombinací algoritmů do celého procesu

Výsledky měření kombinací implementovaných algoritmů pouze potvrzují hodnoty z předchozích kapitol, naměřené pro každý algoritmus zvlášť. Výběr nejrychlejších a nejzajímavějších naměřených hodnot reprezentují tabulky 1-4.

Každá z těchto tabulek představuje výběr měření kombinací, seřazený podle času potřebného ke zpracování složitějšího předloženého obrazu na uvedené kombinaci platformy a procesoru. Zobrazují časy prvních deseti nejrychlejších kombinací, a dále dvě skupiny hodnot, obsahující výrazný skok v naměřeném čase.

V nejrychlejších deseti hodnotách napříč všemi kombinacemi nelze najít použití žádného filtru, z čehož vyplývá, že se jeho použití pro urychlení práce dalších algoritmů vůbec nehodí. Naopak, ze všech provedených měření neexistuje ani jedna změřená kombinace používající nějaký filtr, která by byla rychlejší než nějaká kombinace, pracující bez filtru.

Většinovým zástupcem datových struktur mezi prvními deseti kombinacemi je struktura `BitObraz`, nicméně se zde objevují i kombinace využívající i zbývající dvě struktury. Z toho vyplývá, že jsou rychlosti všech datových struktur velice vyrovnané.

Mezi vyhledávající algoritmy ve všech čtyřech prvních desítkách naměřených hodnot se dostali zástupci téměř všech testovaných algoritmů, avšak dominantními jsou zde především Theo Pavlidis a Square Tracing algoritmus, což potvrzuje výsledky měření popsaných v kapitole 7.7.

Ze zjednodušovacích algoritmů se mezi nejrychlejšími kombinacemi objevují také zástupci všech algoritmů. Nejčastěji je to však nepoužití žádného zjednodušení, ponechání x-tého bodu, počítání vzdálenosti od strany a Reumann-Witkamův algoritmus.

První výrazný skok ve výkonech kombinací algoritmů má na svědomí aplikace filtru. Použití filtru je oproti zbytku procesu natolik časově náročné, že v závislosti na sledované platformě, procesoru a složitosti obrazu způsobí dvou až desetinásobné zpomalení celého zpracování, oproti nejpomalejší kombinaci nevyužívající filtr. Druhý výrazný skok v naměřených hodnotách je také způsoben na všech platformách stejnou příčinou. Tentokrát se jedná aplikaci mediánového filtru na strukturu `BuffObraz`, zapouzdřující obrazové struktury z použitých SDK. Pravděpodobnou příčinou tohoto jevu je nakumulované zpomalení způsobené velkým množstvím přístupů do obrazu, způsobem popsaným na konci kapitoly 7.7.

Kompletní sady naměřených výsledů jsou k dispozici na přiloženém DVD.

Tabulka 1 – Výběr naměřených hodnot časů kombinací na platformě Java a procesoru Core i3

#.	Analýza (datová struktura, filtr, hledání, zjednodušení)	Java i3, čas (ns), jednodušší obraz	Rozdíl %	Java i3, čas (ns), složitější obraz	Rozdíl %
1.	BitObraz, žádný filtr, Theo Pavlidis, žádné zjednodušení	5 011 765	0,0%	7 528 961	0,0%
2.	BitObraz, žádný filtr, Theo Pavlidis, ponechání x-tého	5 113 230	2,0%	8 142 880	8,2%
3.	BitObraz, žádný filtr, Theo Pavlidis, Reumann-Witkam	5 159 724	3,0%	8 514 233	13,1%
4.	BitObraz, žádný filtr, Square Tracing, žádné zjednodušení	5 946 392	18,6%	8 550 280	13,6%
5.	ByteObraz, žádný filtr, Theo Pavlidis, žádné zjednodušení	5 512 698	10,0%	8 558 121	13,7%
6.	BitObraz, žádný filtr, Theo Pavlidis, vzdálenost od strany	5 232 762	4,4%	8 567 797	13,8%
7.	BitObraz, žádný filtr, Theo Pavlidis, Douglas-Peucker	5 127 849	2,3%	8 769 223	16,5%
8.	BitObraz, žádný filtr, Theo Pavlidis, Lang	5 172 273	3,2%	8 829 726	17,3%
9.	BitObraz, žádný filtr, Square Tracing, ponechání x-tého bodu	6 049 144	20,7%	8 988 036	19,4%
10.	ByteObraz, žádný filtr, Square Tracing, žádné zjednodušení	6 597 721	31,6%	9 286 583	23,3%
71.	BuffObraz, žádný filtr, Moore-Neighbor, Douglas-Peucker	12 654 094	152,5%	20 076 711	166,7%
72.	BuffObraz, žádný filtr, Marchin Squares, Douglas-Peucker	12 322 101	145,9%	20 156 540	167,7%
73.	BuffObraz, Gaussův filtr 1, Theo Pavlidis, Douglas-Peucker	26 852 261	435,8%	28 531 173	279,0%
74.	BuffObraz, Gaussův filtr 2, Theo Pavlidis, ponechání x-tého	26 419 256	427,1%	28 679 350	280,9%
75.	BuffObraz, Gaussův filtr 2, Theo Pavlidis, Douglas-Peucker	26 511 975	429,0%	28 759 963	282,0%
263.	ByteObraz, mediánový filtr, Moore-Neighbor, vzdálenost od strany	65 778 718	1 212,5%	70 215 109	832,6%
264.	ByteObraz, mediánový filtr, Moore-Neighbor, Lang	66 956 422	1 236,0%	70 663 619	838,6%
265.	BuffObraz, mediánový filtr, Theo Pavlidis, vzdálenost od strany	114 562 857	2 185,9%	111 856 567	1 385,7%
266.	BuffObraz, mediánový filtr, Theo Pavlidis, ponechání x-tého	116 131 075	2 217,2%	112 567 244	1 395,1%
267.	BuffObraz, mediánový filtr, Theo Pavlidis, Douglas-Peucker	115 590 312	2 206,4%	112 666 126	1 396,4%

Tabulka 2 – Výběr naměřených hodnot časů kombinací na platformě Java a procesoru Celeron

#.	Analýza (datová struktura, filtr, hledání, zjednodušení)	Java Celeron, čas (ns), jednodušší obraz	Rozdíl %	Java Celeron, čas (ns), složitější obraz	Rozdíl %
1.	BitObraz, žádný filtr, Square Tracing, žádné zjednodušení	98 983 757	0,0%	141 475 677	0,0%
2.	BitObraz, žádný filtr, Square Tracing, ponechání x-tého	86 718 747	-12,4%	156 724 912	10,8%
3.	BitObraz, žádný filtr, Square Tracing, Reumann-Witkam	78 055 488	-21,1%	158 537 165	12,1%
4.	BitObraz, žádný filtr, Square Tracing, Lang	104 131 341	5,2%	160 030 648	13,1%
5.	BitObraz, žádný filtr, Theo Pavlidis, vzdálenost od strany	86 915 050	-12,2%	160 155 422	13,2%
6.	BitObraz, žádný filtr, Theo Pavlidis, ponechání x-tého bodu	93 187 523	-5,9%	165 322 384	16,9%
7.	BitObraz, žádný filtr, Square Tracing, Douglas-Peucker	102 860 815	3,9%	165 605 459	17,1%
8.	BitObraz, žádný filtr, Square Tracing, vzdálenost od strany	102 375 966	3,4%	169 483 418	19,8%
9.	BitObraz, žádný filtr, Theo Pavlidis, žádné zjednodušení	81 299 464	-17,9%	174 671 378	23,5%
10.	BitObraz, žádný filtr, Theo Pavlidis, Lang	86 718 768	-12,4%	174 908 298	23,6%
71.	BuffObraz, žádný filtr, Moore-Neighbor, Reumann-Witkam	197 109 517	99,1%	345 557 078	144,3%
72.	BuffObraz, žádný filtr, Moore-Neighbor, Lang	203 904 790	106,0%	346 340 405	144,8%
73.	BuffObraz, Gaussův filtr 2, Square Tracing, Lang	393 735 769	297,8%	421 600 623	198,0%
74.	BuffObraz, Gaussův filtr 1, Theo Pavlidis, ponechání x-tého	385 492 428	289,5%	429 928 826	203,9%
75.	BuffObraz, Gaussův filtr 2, Theo Pavlidis, Lang	377 609 501	281,5%	430 207 394	204,1%
263.	ByteObraz, Gaussův filtr 2, Moore-Neighbor, žádné zjednodušení	950 343 054	860,1%	1 028 852 451	627,2%
264.	BitObraz, Gaussův filtr 2, Moore-Neighbor, vzdálenost od strany	813 785 778	722,1%	1 079 062 700	662,7%
265.	BuffObraz, mediánový filtr, Theo Pavlidis, Lang	1 957 661 094	1 877,8%	1 857 576 099	1 213,0%
266.	BuffObraz, mediánový filtr, Square Tracing, Lang	1 976 943 580	1 897,2%	1 858 699 308	1 213,8%
267.	BuffObraz, mediánový filtr, Square Tracing, Reumann-Witkam	1 973 308 201	1 893,6%	1 861 428 915	1 215,7%

Tabulka 3 – Výběr naměřených hodnot časů kombinací na platformě Qt a procesoru Core i3

#.	Analýza (datová struktura, filtr, hledání, zjednodušení)	Qt i3, čas (ns), jednodušší obraz	Rozdíl %	Qt i3, čas (ns), složitější obraz	Rozdíl %
1.	BitObraz, žádný filtr, Theo Pavlidis, žádné zjednodušení	16 975 538	0,0%	21 382 428	0,0%
2.	BitObraz, žádný filtr, Theo Pavlidis, vzdálenost od strany	20 723 728	22,1%	22 060 920	3,2%
3.	BuffObraz, žádný filtr, Theo Pavlidis, ponechání x-tého	19 228 091	13,3%	22 244 046	4,0%
4.	BitObraz, žádný filtr, Theo Pavlidis, ponechání x-tého	16 656 870	-1,9%	22 445 073	5,0%
5.	ByteObraz, žádný filtr, Theo Pavlidis, žádné zjednodušení	18 770 085	10,6%	22 557 795	5,5%
6.	BitObraz, žádný filtr, Theo Pavlidis, Reumann-Witkam	18 914 091	11,4%	23 100 033	8,0%
7.	ByteObraz, žádný filtr, Square Tracing, žádné zjednodušení	16 252 624	-4,3%	23 530 948	10,0%
8.	ByteObraz, žádný filtr, Theo Pavlidis, Reumann-Witkam	27 462 896	61,8%	23 827 519	11,4%
9.	ByteObraz, žádný filtr, Theo Pavlidis, ponechání x-tého bodu	26 731 741	57,5%	24 014 201	12,3%
10.	BitObraz, žádný filtr, Theo Pavlidis, Douglas-Peucker	19 152 479	12,8%	24 223 469	13,3%
71.	BuffObraz, žádný filtr, Moore-Neighbor, vzdálenost od strany	27 620 827	62,7%	75 862 636	254,8%
72.	BuffObraz, žádný filtr, Moore-Neighbor, Lang	26 324 065	55,1%	88 636 339	314,5%
73.	ByteObraz, mediánový filtr, Theo Pavlidis, Reumann-Witkam	169 793 173	900,2%	232 215 778	986,0%
74.	ByteObraz, mediánový filtr, Marching Squares, Douglas-Peucker	173 886 478	924,3%	237 030 973	1 008,5%
75.	ByteObraz, mediánový filtr, Square Tracing, žádné zjednodušení	181 343 828	968,3%	238 788 719	1 016,8%
215.	BitObraz Gauss2 Moore ZadneZjednoduseni	220 864 231	1 201,1%	353 553 890	1 553,5%
216.	BitObraz Gauss1 Moore Lang(n=25, h=25)	209 833 916	1 136,1%	369 571 825	1 628,4%
217.	BuffObraz Median SquareTracing Lang(n=25, h=25)	223 802 824	1 218,4%	542 707 437	2 438,1%
218.	BuffObraz Median TheoPavlidis VzdalenostOdStrany(25)	225 798 415	1 230,1%	543 787 557	2 443,2%
219.	BuffObraz Median SquareTracing PonechaniX(25)	233 486 824	1 275,4%	545 194 956	2 449,7%

Tabulka 4 – Výběr naměřených hodnot časů kombinací na platformě Qt a procesoru Celeron

#.	Analýza (datová struktura, filtr, hledání, zjednodušení)	Qt Celeron, čas (ns), Jednodušší obraz	Rozdíl %	Qt Celeron, čas (ns), Složitější obraz	Rozdíl %
1.	BitObraz, žádný filtr, Square Tracing, žádné zjednodušení	97 434 262	0,0%	150 121 121	0,0%
2.	BuffObraz, žádný filtr, Theo Pavlidis, Douglas-Peucker	102 331 084	5,0%	162 837 761	8,5%
3.	BitObraz, žádný filtr, Theo Pavlidis, žádné zjednodušení	86 686 566	-11,0%	164 876 075	9,8%
4.	ByteObraz, žádný filtr, Theo Pavlidis, Reumann-Witkam	101 665 562	4,3%	182 038 025	21,3%
5.	ByteObraz, žádný filtr, Square Tracing, žádné zjednodušení	101 926 840	4,6%	210 003 346	39,9%
6.	BuffObraz, žádný filtr, Marching Squares, žádné zjednodušení	100 573 514	3,2%	215 355 127	43,5%
7.	BitObraz, žádný filtr, Theo Pavlidis, Reumann-Witkam	97 822 996	0,4%	216 816 359	44,4%
8.	BuffObraz, žádný filtr, Theo Pavlidis, vzdálenost od strany	102 495 297	5,2%	229 280 033	52,7%
9.	ByteObraz, žádný filtr, Marching Squares, žádné zjednodušení	101 250 243	3,9%	229 712 194	53,0%
10.	BuffObraz, žádný filtr, Theo Pavlidis, Lang	102 682 965	5,4%	237 293 636	58,1%
70.	ByteObraz, žádný filtr, Moore-Neighbor, Douglas-Peucker	120 757 174	23,9%	902 291 447	501,0%
71.	BuffObraz, žádný filtr, Moore-Neighbor, Lang	131 245 361	34,7%	937 109 143	524,2%
72.	BuffObraz, žádný filtr, Moore-Neighbor, vzdálenost od strany	153 015 694	57,0%	1 090 807 274	626,6%
73.	ByteObraz, mediánový filtr, Theo Pavlidis, Reumann-Witkam	919 793 819	844,0%	1 441 613 800	860,3%
74.	ByteObraz, mediánový filtr, Square Tracing, žádné zjednodušení	1 076 274 720	1 004,6%	1 462 702 015	874,3%
215.	BitObraz, Gaussův filtr 2, Moore-Neighbor, vzdálenost od strany	939 826 310	864,6%	2 373 378 285	1 481,0%
216.	BitObraz, Gaussův filtr 2, Moore-Neighbor, Lang	952 482 522	877,6%	2 459 386 462	1 538,3%
217.	BuffObraz, mediánový filtr, Marching Squares, Reumann-Witkam	1 167 649 072	1 098,4%	3 978 082 528	2 549,9%
218.	BuffObraz, mediánový filtr, Theo Pavlidis, Lang	1 161 562 703	1 092,2%	3 993 536 165	2 560,2%
219.	BuffObraz, mediánový filtr, Theo Pavlidis, Reumann-Witkam	1 147 547 959	1 077,8%	3 995 640 325	2 561,6%

7.10 Vyplývající doporučení

Na základě výše uvedených faktů nelze jednoznačně určit pouze jedinou kombinaci struktury a algoritmů, která by všechny ostatní převyšovala. Vzhledem k pomalé aplikaci jakéhokoli filtru lze doporučit vyřazení filtrování obrazu z celého procesu, neboť způsobuje jeho několikanásobné zpomalení.

Vhodná volba algoritmu vyhledávajícího kontury v obraze má přinejmenším dva kandidáty, a to algoritmy Tracing Squares a Theo Pavlidis. Oba dosahují přibližně stejných výsledků a volba ani jednoho z nich nebude špatná.

Složitější situace nastává mezi zjednodušovacími algoritmy, u kterých nemalou roli hraje subjektivní potřeba přesnosti výsledků, kterou do měření prakticky nelze zahrnout. Obecně však platí, že čas potřebný ke zjednodušení polygonů bude přímo úměrný požadavku na přesnost. Nejrychlejšími algoritmy v této skupině jsou algoritmy ponechání x-tého bodu a algoritmus pracující se vzdáleností bodů od strany, tedy algoritmy pracující pouze s lokálním okolím bodu. Nejrychlejším algoritmem zohledňujícím tvar původní linie je Reumann-Witkamův algoritmus.

Z hlediska volby programovacího jazyka volba také není jednoznačná. Obě dvě varianty mají své výhody a nevýhody, přičemž rychlost zpracování je srovnatelná. Výhodou i nevýhodou volby platformy Java je nepochybně odpadající správa paměti. Z hlediska programátora je totiž jednodušší a bezpečnější přenechat správu operační paměti na garbage collectoru. Nevýhodou je to z pohledu zvýšení režie běhu programu a také vyšším množstvím alokované paměti. O implementaci v C++ platí stejná tvrzení opačně. Nevýhodou jsou zvýšené požadavky na dovednosti programátora, kvůli nutnosti správy alokované paměti na úrovni zdrojového kódu. Výhodou je menší množství paměti potřebné pro běh programu a odpadající režie na její správu za běhu programu.

Postup zpracování meteorologické informace s vynecháním filtrování lze doporučit i k nasazení v systémech s omezenou výpočetní kapacitou, neboť nejrychlejší kombinace algoritmů zpracované na slabším stroji podávají výsledky na předloženém složitém obraze již po necelých dvou desetinách sekundy, nezávisle na zvoleném programovacím jazyce. Pro nasazení na takových systémech se pak nabízí další možnosti optimalizace, jakým by mohlo být např. snížení rozlišení zpracovaného obrazu ve fázi sběru dat v otáčce.

Všechny datové struktury poskytují velice vyrovnané rychlosti v přístupu k obrazovým bodům. Z toho je zřejmé, že nasazení paměťově neefektivní struktury `ByteObraz` nepřináší žádné zaznamenatelné zrychlení přístupu díky přímé adresaci a jako jediná tedy není vhodná k reprezentaci obrazu.

Závěr

Postup zpracování obrazu srážkové situace, tak jak je popsán v úvodu kapitoly 3, byl navržen v prostředí firmy T – CZ, a. s. Jako prvotní řešení daného problému byl tento postup implementován kombinací aplikace průměrovacího filtru o rozměrech matice 5x5, k vyhledání srážkových objektů byl zvolen algoritmus Marching Squares (viz kapitola 3.3.5) a k jejich následnému zjednodušení algoritmus Dougles-Peucker (viz kapitola 3.4.5).

Navržený postup byl implementován jako multiplatformní aplikace na platformě Java a v době psaní této práce je nasazen v testovacím provozu s reálnými daty v areálu firmy T – CZ, a. s. v Pardubicích. Aplikace je koncipována jako desktopová, běžící na samostatném hardwaru architektury x86, přičemž aplikace splňuje požadavky nastolené v kapitole 4.

Vzhledem k tomu, že prvotní řešení bylo vytvořeno bez předchozí hlubší analýzy, rozhodl jsem se s podporou firmy vypracovat tuto práci, na jejímž konci by mělo být možné potvrdit či vyvrátit vhodnost prvotně použitých algoritmů. Z toho důvodu jsem k řešení této úlohy přistoupil obecněji, s cílem prozkoumání možností více algoritmů a jejich vzájemných kombinací.

Cíl práce se podařilo naplnit a z kapitoly 3 vyplývá, že původní nastavení procesu nebylo z hlediska minimalizace času, potřebného ke zpracování, optimální. Všechny poznatky získané zpracováním této práce budou předány ke zvážení vedení společnosti a bude nastolena diskuze o jejich využití při budoucí aktualizaci dosud používané aplikace. Změny by měly spočívat především k vyřazení filtrování obrazu, neboť se v této práci podařilo dokázat, že celý proces namísto zamýšleného zrychlení, výrazně zpomalí. Své zrychlení však přinese i implementace jiných algoritmů pro identifikaci srážek a zjednodušení jejich kontur.

Literatura

[1] BEZOUŠEK, Pavel a Pavel ŠEDIVÝ. *Radarová technika*. Praha: Vydavatelství ČVUT, 2004. ISBN 80-01-03036-9.

[2] ŠEBESTA, Jiří. *Radiolokace a radionavigace*. Brno: Vysoké učení technické v Brně, 2004. ISBN 80-214-2482-6.

[3] Single European Sky. EUROCONTROL. *EUROCONTROL* [online]. 2010 [cit. 2013-05-12]. Dostupné z: http://www.eurocontrol.int/dossiers/single-european-sky

[4] Air traffic control. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-13]. Dostupné z: http://en.wikipedia.org/wiki/Air_traffic_control

[5] Single Sky Support. EUROCONTROL. *EUROCONTROL* [online]. 2010 [cit. 2013-05-12]. Dostupné z: http://www.eurocontrol.int/articles/single-sky-support

[6] Transmission of Monoradar Derived Weather Information. EUROCONTROL. *EUROCONTROL* [online]. 2010 [cit. 2013-04-15]. Dostupné z: http://www.eurocontrol.int/sites/default/files/content/documents/nm/asterix/cat008-asterix-monoradar-weather-data-part-3.pdf

[7] ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika*. Brno: Computer Press, a.s., 2004. ISBN 80-251-0454-0.

[8] Řízení letového provozu: Jednotné evropské nebe - Single European Sky. ŘÍZENÍ LETOVÉHO PROVOZU ČESKÉ REPUBLIKY. *Řízení letového provozu České republiky* [online]. 2002 [cit. 2013-04-16]. Dostupné z: http://www.rlp.cz/generate_page.php?page_id=839

[9] Filtrace obrazu. *Odbor termomechaniky a techniky prostředí* [online]. [cit. 2013-05-02]. Dostupné z: http://ottp.fme.vutbr.cz/~pavelek/optika/1506.htm

[10] GHUNEIM, Abeer. Contour Tracing. *Contour Tracing* [online]. [cit. 2013-05-10]. Dostupné z: http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/index.html

[11] Marching squares. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-01]. Dostupné z: http://en.wikipedia.org/wiki/Marching_squares

[12] *Algoritmy v digitální kartografii*. Praha: Karolinum, 2008. ISBN 9788024614991.

[13] Generalizace: Přednáška z předmětu Tematická kartografie. ČERBA, Otakar. ZÁPADOČESKÁ UNIVERZITA. *Generalizace* [online]. 2004, 2011-08-28 [cit. 2013-04-20]. Dostupné z: http://gis.zcu.cz/studium/tka/Slides/generalizace.pdf

[14] HRNČÍŘOVÁ, Jana. *Algoritmy geometrické generalizace pro PostGIS*. Praha, 2009. Diplomová práce. ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE. Vedoucí práce Jiří Cajthaml.

[15] Kartografické generalizacní algoritmy. BAYER, Tomáš. *Přírodovědecká fakulta UK v Praze* [online]. 8888 [cit. 2013-04-24]. Dostupné z: http://web.natur.cuni.cz/~bayertom/Adk/adk8.pdf

[16] Class BufferedImage. ORACLE. *Oracle Documentation* [online]. 2011 [cit. 2013-04-27]. Dostupné z: http://docs.oracle.com/javase/6/docs/api/java/awt/image/BufferedImage.html

[17] QImage Class Reference. *Qt Project* [online]. [cit. 2013-04-27]. Dostupné z: http://qt-project.org/doc/qt-4.8/qimage.html

Příloha A – Ukázka zpracování obrazu (Mediánový filtr, Moore-Neighbor alg., Langův alg.)