VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VEKTOROVÁ ANALÝZA V GIS SYSTÉMECH

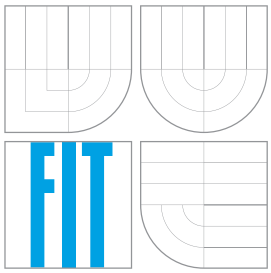
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ ŽIVOTSKÝ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VEKTOROVÁ ANALÝZA V GIS SYSTÉMECH

VECTOR ANALYSIS IN GIS SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ ŽIVOTSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN HRUBÝ, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce pojednává o geografických informačních systémech s důrazem na tzv. vektorovou analýzu. Cílem práce je vytvořit knihovnu, která implementuje vnitřní reprezentaci geografických dat a provádí nad nimi analytické operace. Práce popisuje logickou strukturu aplikace a hierarchii abstraktních tříd. Formou pseudokódů je předveden soubor jednotlivých výpočetních operací. Práce silně vychází z geografického systému GRASS. Na tento GIS je napojitelná formou importu a exportu vektorových dat ve formátu shapefile. Koncepce ukládání vrstev je navržena pro snadnou manipulaci s daty a jejich přenositelnost. Hlavní myšlenkou aplikace je rozčlenění knihovny do jednotlivých modulů. Tyto moduly jsou na sobě nezávislé, zároveň však mohou využívat navzájem své funkce. Závěrem jsou předvedeny implementované vektorové analýzy a reprezentace jejich výpočtu.

Abstract

This master's thesis deals with the geographic information systems with an emphasis on so - called vector analysis. The aim of the thesis is to create a library that implements the internal representation of geographic data and performs analytical operation over them. The thesis describes the logical structure of the application and the hierarchy of abstract classes. In the form of pseudo-code is shown a set of individual computing operations. Work is strongly based on geographic system GRASS. This GIS is connectable through import and export vector data in shapefile format. Concept of storing layers is designed for ease of data manipulation and their portability. The main idea behind the application is to split the library into individual modules. These modules are independent, but also can benefit from each other's functions. Finally, as demonstrated implemented vector analysis and representation of their calculation.

Klíčová slova

GIS, vektor, analýza, geografický, informační, systém, bod, linie, polygon, buffer, překryv, síť

Keywords

GIS, vector, analysis, geographic, information, system, point, line, polygon, buffer, overlay, net

Citace

Tomáš Životský: Vektorová analýza v GIS systémech, diplomová práce, Brno, FIT VUT v Brně, 2015

Vektorová analýza v GIS systémech

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Martina Hrubého. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Životský
26. května 2015

Poděkování

Děkuji panu Maritnu Hrubému za vedení mé diplomové práce a za jeho veškeré odborné konzultace a cenné rady.

Obsah

Kapitola 1

Úvod

V průřezu celých dějin lidstva se setkáváme s potřebou zaznamenávat naše okolí. Ať už se jednalo o plány bitev, objevitelské mapy, plány vesnic a měst, stále se jedná o objekty reálného světa, které si lidé vkládali do nějaké imaginární struktury.

S narůstajícím objemem geografických dat byla potřeba tyto informace smysluplně ukládat. To dalo vzniknout mnoha vědeckým disciplínám, například kartografii a geografii. Vynalezení podpůrných nástrojů, jako je kompas, značně zjednodušilo proces mapování. Tím mohly začít vznikat větší a přesnější mapy.

Nejdříve se používalo pro zaznamenání mapy pergamenu, látky a papíru. Výhodou pro tyto materiály je snadná manipulace a transport. Naneštěstí tyto materiály rychle podléhají fyzickému opotřebení, ať už působením vnějších vlivů nebo samotnou manipulací s nimi. Velkou nevýhodou papírových map byla hlavně jejich náročná výroba a distribuce. Ještě dnes, i když je k dispozici technologie pro snadné kopírování, je výroba nových map finančně náročná. Krajina se navíc neustále mění, staví se nové budovy, silnice, rozšiřují se zemědělské pozemky atd. Není proto možné každou změnu reflektovat vydáním nové mapy. Jedním z používaných řešení je pravidelné vydávání aktualizované verze mapy. Změny se tak promítnou shlukově. Nevýhodou této metody je, že do vytvoření nové verze jsou k dispozici neplatná data. Další nevýhodou je malý projekční prostor mapy. Informace, které chceme z mapy vyčíst, mohou být velmi rozdílného charakteru. Papírová mapa má omezenou zobrazovací kapacitu, a tak pro zobrazení kompletní informace může být zapotřebí více map. Tyto mapy se dle typu informací dělí na topografické, tématické, katastrální atd.

V současnosti, s rozvojem technologie a počítačů, je systém papírových map nahrazován digitalizovanou formou. Vzniká geografický počítačový software, tzv. *geografické informační systémy (GIS)*. Se zaváděním GISů se pojí celá řada komplikací, nicméně výhody takových systémů značně převyšují formu stávajících map. Díky digitalizaci odpadají zmíněné problémy s opotřebováváním materiálu. Zároveň aktualizace map je velmi snadná, neboť se upraví pouze počítačový soubor. Velkou výhodou oproti papírovým mapám je uložení informací o mapovaném objektu na jednom místě. Například vlastnosti silnic jako název, stav, třída, rychlostní limit atd. jsou uloženy jako atributy konkrétního objektu. Pokud poté chce uživatel zjistit nějakou z požadovaných informací, zobrazí si pouze tyto atributy. S použitím GIS se pojí analýza dat. Protože manuální provádění vzdálenostních, překryvových či síťových operací je dosti časově i výpočetně náročné, GIS jsou ideálním nástrojem pro jejich automatizaci.

Geografické informační systémy lze rozdělit dle typu zpracování dat na rastrové a vektorové. Oba přístupy mají své využití. Tato práce se však zabývá pouze vektorovou analýzou. Pro některé analýzy je zapotřebí výslovně vektorového subsystému. Do této skupiny se řadí

například síťové analýzy. V těchto případech je nutná informace o topologických sousedech a rozdělení sítě na subsystémy. Podstatou vektorových GIS je rozdělení mapy do jednotlivých geografických objektů, které odpovídají reálným strukturám. Každý z těchto objektů má svoje souřadnice, případně délku, výšku, hranice, ... Nevýhoda vektorových systémů je v jejich vnitřní interpretaci. Aby se mohlo s nějakým geografickým objektem pracovat, je nutno jej uložit do vnitřní struktury a navázat ho na ostatní objekty. V případě hustého členění mapového podkladu tak může vzniknout velká množina objektů.

Mnoho analytických operací může být provedeno jak nad rastrovou, tak nad vektorovou formou dat. Takovými operacemi jsou například výběrové dotazy dle polohy, dle atributů, vzdálenostní analýzy či překryvové analýzy. Naopak čistě vektorovou operací jsou zmíněné síťové analýzy.

1.1 Cíl práce

Tato diplomová práce čtenáře seznámí s principy fungování geografických informačních systémů, způsoby uložení vektorových dat a s jejich manipulací. Dále bude předvedeno několik algoritmů pro vektorovou analýzu.

Výstupem diplomové práce je knihovna La'GIS, která implementuje subsystém pro uložení vektorových dat a algoritmy vektorové analýzy. Tato knihovna je snadno napojitelná do většího geografického informačního systému, například jako součást systému GRASS.

Knihovna je rozčleněna do modulů. To sebou přináší výhody vzájemné nezávislosti. Zároveň je však možné v jednom modulu volat funkce modulu jiného, takže nejsou tytéž operace implementované na více místech. Příkladem je knihovní funkce pro výpočet vzdálenostní analýzy, která využívá operací modulu pro překryv vrstev.

Krom knihovny La'GIS výsledná aplikace obsahuje i sadu spustitelných demo - programů, které tvoří uživatelské rozhraní pro vektorovou analýzu.

1.2 Struktura dokumentu

V tomto odstavci je nejstručnější popis toho, co v jaké pasáži lze najít a čemu byla při tvorbě dokumentu věnována zvláštní pozornost.

Kapitola 2 pojednává o historickém i současném využití geografických informačních systémů. Úvodním slovem je definován pojem GIS a co takový systém obsahuje. Stručně je vysvětlena problematika geodézie a principy modelování zemského povrchu.

Čtenář je seznámen s typy vektorových objektů v kapitole 2.4. Kapitola 2.6 popisuje způsob organizace vektorových dat a uložení do jednotlivých vrstev. Následuje stručný popis jednotlivých, v současnosti užívaných GIS a jejich krátká charakteristika 2.7.

Velmi důležitou součástí vektorových geografických systémů je informace o topologii objektů, což je popsáno v kapitole 2.5.

V kapitole 3 je popsán koncept knihovny La'GIS. Je zde znázorněno rozdělení do logických celků a závislosti jednotlivých modulů. Současně jsou v této kapitole popsány pseudokódy pro operace vektorové analýzy a jejich rozbor. Čtenář je zde seznámen s vnitřním formátem vektorových dat. Taktéž je zde popsáno uživatelské a programátorské rozhraní pro knihovnu La'GIS.

Kapitola 4 popisuje způsob řešení jednotlivých implementačních problémů, například způsob a formát ukládání dat. Tato kapitola taktéž popisuje způsob napojení knihovny na větší GIS.

Závěrečná kapitola 5 se zabývá testováním aplikace na sadě vektorových vrstev, kombinování jednotlivých algoritmů a zhodnocení časů jejich výpočtů.

Kapitola 2

Geografické informační systémy

Výstavba GIS a digitalizace mapových podkladů je v současnosti velmi populární téma. Pro svoji činnost využívají geografických systémů stavební firmy, statistický úřad, stavební firmy, správa silnic, vodohospodářský úřad a podobně.

V této kapitole budou vysvětleny základní principy GIS, problémy, se kterými se potýkají, a uvedeno několik významných zástupců a jejich základní charakteristika.

Existuje více definic geografického systému [13], nicméně všechny se shodují v tom, že v současnosti se za GIS považuje obecně počítačový systém, který uchovává a zpracovává geografická data. Krom softwaru a hardwaru se do systému řadí i veškeré ostatní nástroje, které slouží pro vstup a výstup geografických map či komunikaci mezi rozličnými prvky [8].

První zmínky o užití výpočetní techniky k řešení problémů spojených s prostorovými daty lze patrně hledat ve vojenském odvětví na přelomu padesátých a šedesátých let. Kromě vládních institucí se vývojem GIS zabývali spíše akademičtí pracovníci a významné vědecké osobnosti tehdejší doby. V civilním odvětví se použití GIS objevilo zhruba o dvě desetiletí později, v sedmdesátých až osmdesátých letech. První vážné pokusy zpracovat velké množství prostorových dat prováděla společnost, jež dala později vzniknout *Canada Geographic Informatin System (CGIS)* [15]. Dodnes je CGIS jedním z nejefektivnějších využití prostorových dat a jejich zpracování. I přes nezdar mnoha dalších systémů, především kvůli špatnému návrhu a nepředvídání speciálních technických problémů kvůli obrovskému množství dat, se kterými je třeba pracovat, se užívání GIS firmami stávalo v druhé polovině sedmdesátých let spíše pravidlem než výjimkou. Příkladem takovýchto neúspěšných projektů může být *the Land Use and Natural Resources (LUNR)*. Jelikož dochází k masivnímu nasazování systémů do komerčního prostředí, dochází i k vývoji jednodušších nástrojů pro správu dat. Díky tomuto trendu se postupně přešlo až na vývoj celých GIS orientovaných na práci s laickou veřejností [11].

V současnosti využívá mnoho firem geografické informační systémy například pro výpočet nejvhodnějšího umístění budovy (nákupní střediska, letiště, . . .), pro vyznačení záplavových oblastí či jinak geologicky rizikových oblastí (pojišťovny) a mnoho dalšího. Jedním z kurióznějších případů užití může být využití GIS policejními složkami Metropolian Toronto Police při stíhání podezřelých po městě [12].

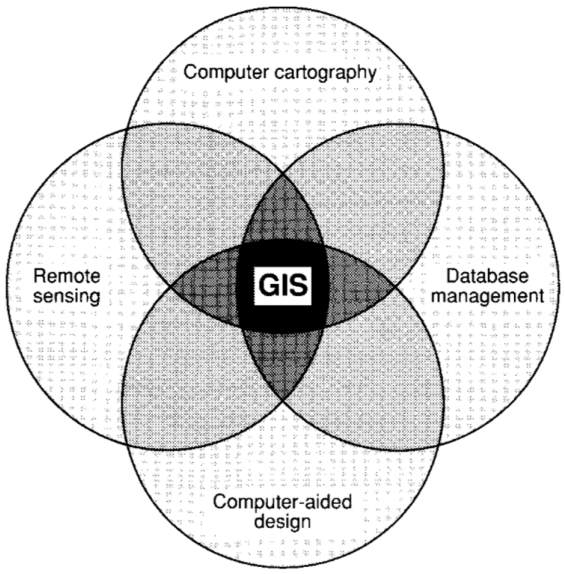
Využití GIS pomáhá efektivně spravovat lesy, různá civilní i vojenská zřízení, petrolejový průmysl, zemědělství, těžařské odvětví či plánování výstavby spojnic mezi městy a státy. Rozvoj GIS je tak jedním z potenciálních směrů, které budou mít v budoucnosti ještě významné uplatnění.

2.1 Principy GIS

Cílem veškerých geografických systémů je zjednodušit a zpřehlednit práci s prostorovými daty a umožnit jejich snadné šíření. Na rozdíl od tištěných map, kde úložiště a reprezentace dat je ve své podstatě jedno a to samé, v GIS se jedné o dvě rozdílné věci. Díky tomu se ta stejná data mohou reprezentovat mnoha způsoby v závislosti na potřebě. Jakmile je podklad uložen v počítači, můžeme pohled přibližovat či oddalovat, vybírat pouhé výseče, měřit vzdálenosti mezi dvěma místy, slučovat části podkladu dle vlastností a jiné. Například při vykreslení silnice, lze snadno zjistit její číslo, povrch, počet pruhů a případně rychlostní limit. Vše záleží jen na obsáhlosti modelovaných dat, nejsme omezeni prezentačním prostorem, jak je tomu v případě map papírových.

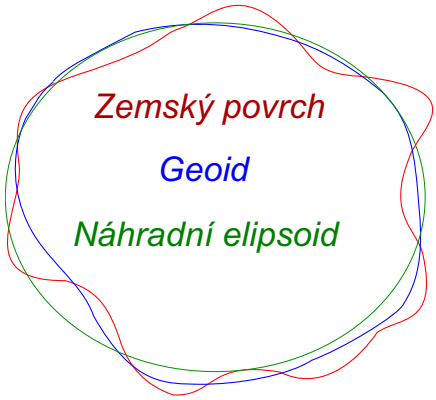
Geografický informační systém se skládá nejen z hardwaru a softwaru, ale i ze speciálních zařízení potřebných pro výrobu mapových výstupů a podobně. Proto se na stavbě GIS podílí mnoho různých vědeckých odvětví. Jejich znázornění je na obrázku 2.1. Každý GIS by tak měl zajišťovat následující funkce nad doloženými daty [8]:

- získávání a ověřování
- kompilace
- úložiště
- aktualizace
- manipulace
- načtení a prezentace
- analýza a kombinace



Obrázek 2.1: Znázornění vztahu mezi jednotlivými obory a GIS [13]

Základem geografického informačního systému je robustní jádro systému, které zajišťuje vnitřní reprezentaci geografických objektů a jejich komunikaci. Tyto objekty jsou obvykle



Obrázek 2.2: Vztah povrchu Země, geoidu a náhradního elipsoidu

drženy v nějaké vnitřní struktuře (seznamu, stromu, asociativním poli, instanci třídy . . .), která reprezentuje mapovou vrstvu.

Velmi důležitým údajem pro vektorová data v GIS je tzv. katalogové číslo. To reprezentuje unikátní označení geografického objektu a slouží jako identifikátor konkrétního objektu.

2.2 Geodetické modely Země

Protože v geografických informačních systémech se pracuje s prostorovými daty, které musí být relevantní ke konkrétnímu místu, je potřeba veškeré souřadnice a prostorové označení vztáhnout k nějaké globální soustavě. Touto problematikou se zabývá vědecký obor *geodézie*.

Zemský povrch není absolutně kulatý. Jak je vidět na obrázku 2.2, pro matematické vyjádření povrchu Země se používá takzvaný *geoid*, případně elipsoid viz odstavec 2.2.1.

Geologická poloha se značí systémem zeměpisné šířky a délky. Každý bod povrchu Země je pak označen dvojicí v rozsahu $\pm 180\,°$pro východní či západní délku a $\pm 90\,°$pro severní nebo západní šířku. Tato dvojice není však úplnou informací pro referování konkrétního bodu na Zemi. Zbývajícími údaji jsou informace o středu soustavy a poloměru osy, která tvoří povrch koule či náhradního elipsoidu.

Pro různé státy se používají různé referenční elipsoidy. Tím, že povrch Země není dokonalá koule, matematické referenční elipsoidy jsou navrženy tak, aby se co nejvíce přimykaly ke geoidu pouze na určitém území. Pokud tak chcete zjistit vzdálenost mezi dvěma různými souřadnými systémy, je třeba s touto závislostí počítat a použít patřičný převod na shodnou soustavu.

2.2.1 Náhradní elipsoidy

Tato sekce pojednává o zavedených náhradních elipsoidech. Hlavní parametry každého elipsoidu je délka hlavní a vedlejší poloosy a jeho střed. Většinou se tyto údaje uvádí jen jako délka hlavní poloosy (př. $a = 6,378,137\,m$) a převrácená hodnota zploštění ($\frac{1}{f} = 298.257223563$).

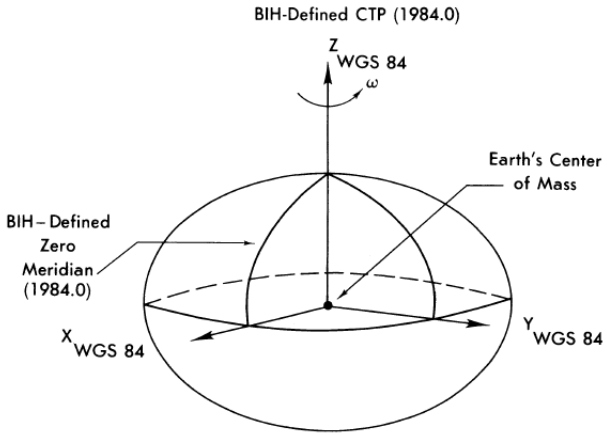
Ať už ale použitý elipsoid má jakékoliv parametry (délka hlavní a vedlejší poloosy, počátek), tak nikdy nebude dostatečně přesně aproximovat reálný povrch po celém svém povrchu. Proto bylo vytvořeno vícero různých elipsoidů, které jsou dostatečně přesné pro část zemského povrchu, nicméně pro jinou část mají znatelně rozdílný tvar a pozici. Tyto

elipsoidy jsou poté používány právě v oblastech, kde se dostatečně přimykají reálnému povrchu (případně geoidu).

V GIS se používá i termín *datum*, což je jiný název pro náhradní elipsoid, ačkoliv literatura [17] uvádí, že mezi těmito dvěma pojmy jsou mírné rozdíly.

Náhradní elipsoid WGS 84

Referenční elipsoid WGS 84 je celosvětově uznávaný geodetický standard. Vyvinulo ho roku 1984 ministerstvo obrany USA. Používá systém zeměpisné šířky a délky s nultým poledníkem nedaleko Greenwich nultého poledníku (5.31 úhlové vteřiny východně). Středem pro tento elipsoid bylo zvoleno těžiště země. Maximální odchylka od geoidu je 60 m.



Obrázek 2.3: Referenční elipsoid WGS 84

Hayfordův náhradní elipsoid

Tento elipsoid je důležitý pro oblast Evropy, kde slouží jako referenční elipsoid pro datum *ED50*. V rámci politických důvodů byl tento elipsoid časem přejmenován na *International Ellipsoid*. Postupem času je však vytlačován elipsoidy jako GRS 80 a WGS 84.

Náhradní elipsoid Clarke 1866

Dalším příkladem referenčního elipsoidu může být *Clarke 1866*, občas značený jako *Clarke66*. Tento elipsoid je základem severoamerického data *NAD27*. Zakladatel elipsoidu Alexander Clarke byl anglické národnosti, proto původní jednotky hlavní a vedlejší osy jsou definovány v britských stopách.

2.2.2 Geografické souřadné systémy

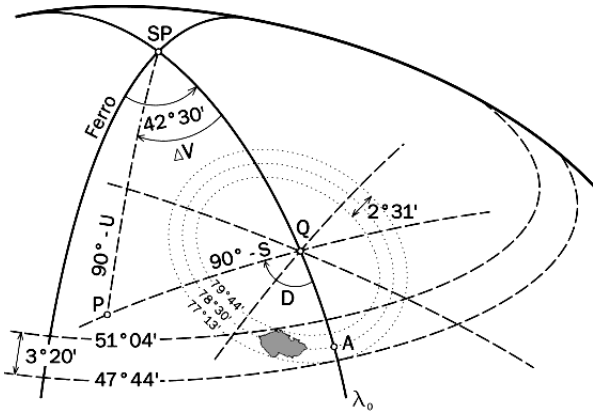
Vyjádřením geologických souřadnic pomocí šířky a délky lze popsat libovolné místo na Zemi (v rámci referenčního elipsoidu). Pro lidské zpracování však tento systém není příliš intuitivní a tak se zavedl systém geografických souřadných systémů. Ty se získávají různou

formou projekce náhradního elipsoidu na válec, kužel atd. Hlavním smyslem těchto projekcí je deformace zemského povrchu do roviny. Tím pádem lze souřadný systém považovat za pravoúhlý a jako jednotku si zvolit například metr.

V následujících odstavcích bude uvedeno několik málo geografických souřadných systémů, se kterými se můžeme v České republice nejčastěji setkat.

Souřadný systém S - JTSK

Souřadný systém Jednotné trigonometrické sítě katastrální navrhl a zpracoval Josef Křovák a vyznačuje se především tím, že je navržen právě pro Českou republiku a Slovensko. Promítá do pravoúhlého souřadného systému, jehož jednotkou je metr, a zároveň není zkreslení způsobené projekcí do roviny nepřijatelně velké. Je založen na konformním kuželovém zobrazení zmenšené Gaussovy koule [2].



Obrázek 2.4: Projekce konformního kužele na Gaussovu kouli

Souřadný systém UTM

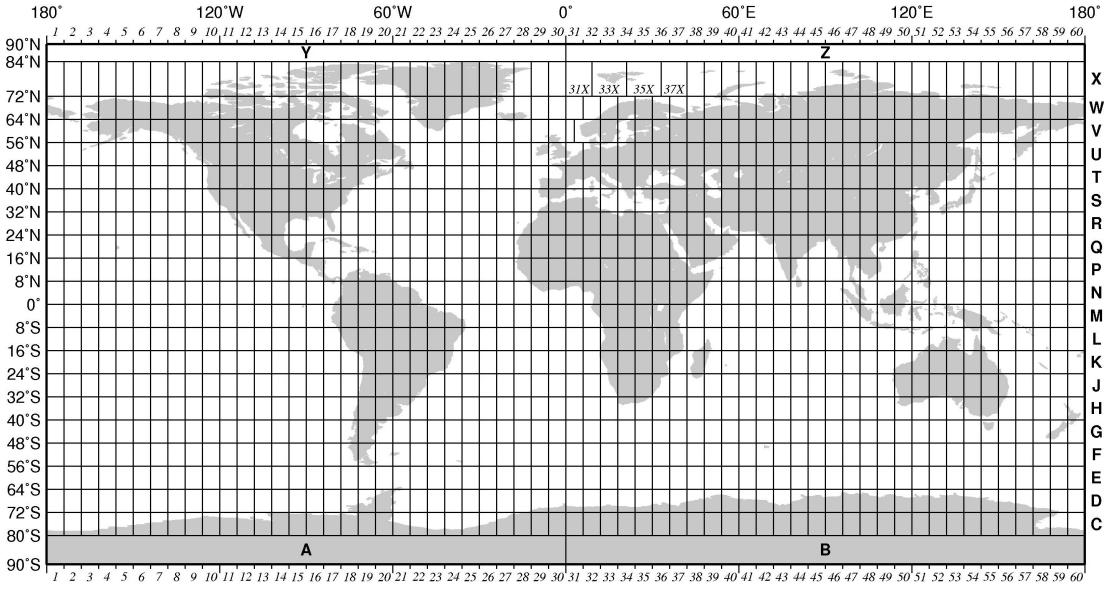
Formát UTM je dalším z formátů, který zobrazuje zaoblený povrch elipsoidu do pravoúhlé soustavy se souřadnicemi v metrech. Princip zobrazení tohoto modelu je, že se povrch elipsoidu rozdělí na pruhy podle jednotlivých rovnoběžek a následně i podle poledníků. Tím vzniká šachovnicová síť 2.5. Nevýhoda tohoto formátu je ta, že nelze měřit vzdálenosti a provádět výpočty mezi body rozdílných segmentů. Každá souřadnice je pak složena ze tří čísel, udávající x - souřadnici, y - souřadnici a číslo segmentu 2.5.

2.2.3 Výpočet vzdálenosti

Ať už používáme kterýkoliv elipsoid a projekci, vzdálenost dvou bodů v projekci neodpovídá vzdálenosti odpovídajících bodů ve skutečnosti. Proto je třeba provádět patřičné úpravy výpočtu.

Vzdálenost v kartézském souřadném systému

Jak bylo zmíněno, projekce se snaží transformovat kulovitý souřadný systém do pravoúhlého kartézského souřadného systému s nějak rozumně definovanou jednotkou. Součástí údaje o projekci je i takzvaný *K - faktor* nebo *faktor projekce*, který udává hodnotu, kterou se



Obrázek 2.5: Rozdělení válcové projekce do UTM zón

musí vynásobit reálná vzdálenost, aby se získala vzdálenost v projekci. Jinými slovy se musí projekční vzdálenost vynásobit inverzní hodnotou faktoru, aby se získala reálná vzdálenost.

$$\text{reálná vzdálenost} \cdot \text{k - faktor} = \text{projekční vzdálenost}$$

$$130\,210.44 \cdot 0.999953617 = 130\,204.40$$

Samotný výpočet vzdálenosti dvou bodů pak může být realizovaný například Euklidovskou metrikou:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Vzdálenost na náhradním elipsoidu

Zásadní rozdíl mezi metrickými systémy a úhlovými je ten, že metrické měří vzdálenost objektů v rovině, zato úhlové v zaobleném prostoru. Kdyby tytéž body v úhlovém systému byly počítány Euklivskou metrikou, výsledkem by byla přímá vzdálenost mezi body. V tomto případě je však nutno zohlednit zaoblení elipsoidu, čili se pro výpočet vzdálenosti musí užít metriky jiné [16].

2.3 Formát vektorových dat

Tato kapitola popisuje přístupy k ukládání vektorových dat na disk. Jednotlivé formáty se od sebe mohou radikálně lišit, a přesto musí být prezentovány uživateli naprosto stejně. Každý geografický informační systém má svůj způsob ukládání dat, tzv. nativní formát. Existují různé obecné formáty, které se používají pro sdílení geografických dat. Konkrétní GIS pak podporuje načítání těchto dat v obecném formátu a převádí si je do formátu svého. O geografických datech se ukládá několik informací. První z nich jsou x a y souřadnice a souřadný systém, do kterého spadají. Dále se pak mohou ukládat atributy geografických objektů, prostorový index a podobně.

Jelikož GIS ze své podstaty 2.1 není jen prohlížeč mapových vrstev, ale především nástroj pro analýzu a operace nad těmito daty, je nutno při vybírání alternativy uložení dat myslet nejen na efektivitu čtení, ale hlavně na přehlednost a flexibilitu dat při zápisu nových, či úpravách stávajících dat.

V následujících odstavcích jsou popsány dnes nejběžněji používané způsoby uložení dat a zhodnocení jejich efektivity.

2.3.1 Sekvenční soubor s vektorovými daty

Způsob ukládání dat do sekvenčního souboru je v současnosti velmi rozšířenou variantou úložiště. Může za to vývoj GIS, kdy tento přístup byl na počátku velmi levným a snadno dosažitelným řešením. Tato data mohou být uložena buď formou plain textu, jako tomu je v případě ESRI shapefile, nebo binárně.

V současnosti, s přibývajícími objemy dat a jejich potřebou je často upravovat, se tato metoda nezdá příliš vhodná. Pokud bychom chtěli připojit nová data ke stávajícím, není problém soubor rozšířit o požadovanou velikost. Problém však nastává, když je potřeba upravit data uložená uprostřed souboru, což plyne z fyzického uložení souboru na disku.

Jedním ze standardizovaných formátů je definující jazyk *PostScript* [9]. Soubory obsahující skript v tomto jazyce je ASCII kódován, ale taktéž obsahuje velmi komplexní hlavičku popisující atributy dat. I když je tento jazyk standardizován pro využití v GIS, stále je zapotřebí užití interpretu, neboť se jedná o skriptovací jazyk.

Dalším formátem sekvenčního souboru je například *Shapefile* od firmy ESRI. Funkce `v.in.ogr` a `v.out.ogr` geografického systému GRASS umožňuje import a export tohoto formátu [1]. Formát shapefile (.shp) dokáže uchovávat pouze jeden typ geometrie na jednu vrstvu, což znamená, že adresář definující specifickou vrstvu může obsahovat více souborů, v závislosti na tom, jak jsou data uložena. Například pro polygony soubor `polygons.shp` a pro linie `lines.shp` [7]. Pro efektivnější vyhledávání v souboru existuje od verze OGR 1.10 možnost indexování, ale editování těchto indexů nelze provádět přímo - lze pouze index vytvořit a případně smazat. Pokud by bylo třeba uložit atributy k vektorovým objektům, je třeba vytvořit další soubor, který tyto informace bude obsahovat. Obdobně je to s uložením projekce a podobně. To vede k tomu, že v rámci jedné vrstvy existuje vícero souborů, které jsou potřebné pro její správnou interpretaci.

Geografický informační systém GRASS podporuje i další vektorový formát dat, jako je například *MapInfo* [1]. Na rozdíl od formátu firmy ESRI vnímá MapInfo adresář jako soubor veškerých dat a jednotlivé soubory v něm jako samostatné vrstvy.

Firma Hawlett - Packard zavedla svůj vlastní formát HPGL. Soubor je psán v ASCII znacích a strukturován je řádkováním. Každý bod je na vlastním řádku, po sobě jdoucí řádky pak značí sekvenci bodů - linie. Jednotlivá primitiva jsou od sebe oddělena prázdným řádkem. Tento formát podporuje pouze geometrické vlastnosti objektů, nikoliv jejich topologii [9].

2.3.2 Databázový přístup k vektorovým datům

Počátkem sedmdesátých let začaly vznikat první databázové systémy pro uložení prostorových dat s ohledem na operace nad nimi. Největší přínos a důvod pro jejich zavedení bylo zmíněné upravování dat, bezproblémové vkládání nových a především udržení konzistence dat po dlouhou dobu. Z podstaty databázového přístupu byla zavedena množina funkcí, které usnadňovaly a zefektivňovaly operace jako řazení, uspořádávání či vyhledávání v datech.

Jedny z prvních typů databází byly takzvané hierarchické databáze. Modelovaný systém tak byl rozdělen dle úrovně abstrakce na jednotlivé soubory, mezi kterými byly hierarchické vazby.

S postupem času, jak se geografické systémy vyvíjely, se začaly používat pro ukládání dat databázové sklady (počátky sedmdesátých let). *Database management systems (DBMS)* je optimalizovaný software pro ukládání a vyhledávání negrafických atributových dat. Tyto systémy jsou implementovány pro rychlost a jednoduchost, nikoliv pro operace s velkými daty. Taktéž je na nich zavedena hrstka jednoduchých analytických operací [13].

Dalším krokem vývoje byl relační model dat, který se používá i dnes. V závislosti na konkrétním GISu se mohou formy databáze lišit. Databáze tak může být implementována například skupinou souborů, kde každý soubor představuje jednotlivé tabulky databáze. Jiným přístupem pak může být jediný soubor pro celou vrstvu. Jednotlivé tabulky jsou pak součástí toho daného souboru a mohou zaznamenávat geometrii, topologii a indexaci elementů.

2.3.3 Kombinace přístupů k ukládání dat

Častým přístupem současnosti je kombinace sekvenčního a databázového přístupu. Určité atributy a vlastnosti jsou uloženy v relační databázi a odtud jsou přes identifikátor odkazovány do sekvenčního souboru. To způsobuje rozčlenění jedné geografické vrstvy do více souborů, přičemž pro kompletní informaci o datech je třeba mít přístup k většině z nich.

Některé GIS, jako například GRASS, umožňují při ukládání vrstvy explicitně definovat, jakým způsobem se má vrstva uložit. Lze tak vynutit uložení do jiného formátu než nativního.

2.4 Typy vektorových objektů

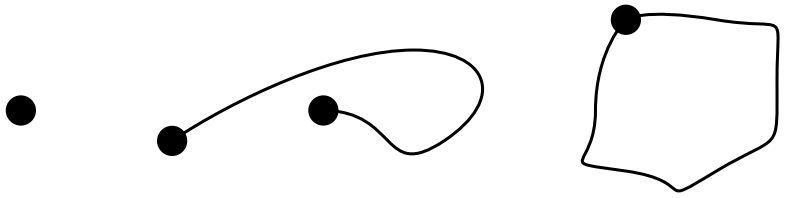
Když se podíváme na výstup nějaké vrstvy, vidíme komplexní reprezentaci uložených dat. Například při vykreslení vrstvy modelující povodí řeky se může zdát, že se jedná o jeden velký celek. Celá tato vrstva je ale vnitřně uložena jako soubor jednotlivých entit, které reprezentují samostatné řeky. Teprve prezentace dat nám z toho vytvoří jedno velké povodí. Vektorové objekty jsou též někdy značeny jako vektorová *primitiva*.

První generace vektorových dat byla modelována pouze za pomocí jednoduchých linií. Simulovalo to tak způsob, jakým by kartograf zanášel údaje do kreslené mapy. Z důvodu, kdy se v souboru s daty muselo programově procházet z místa na místo ve snaze následovat jednotlivé úsečky, byl tento systém označen programátorem Nickem Chrismanem za *špagetový*. Tento název se v oboru uchytil a dnes je znám pod pojmem *Cartographic spaghetti*. I přes zjevnou neefektivitu práce s takovýmto formátem dat, mnoho GIS tento způsob používá. Většina ostatních GIS dokáže podklady v tomto formátu načíst, vnitřně si je ale pak převede na vlastní topologickou reprezentaci [9].

Základní stavební kameny každého geografického systému jsou v podstatě tři primitiva. Každé je pak v rámci vrstvy označeno svým vlastním unikátním identifikátorem [12]. Primitiva na sebe vzájemně navazují, proto je uvažujme postupně, dle vzrůstajícího stupně dimenze 2.6.

- *Bod* - Označuje místo jako průniky linií či jejich krajní body. Každý bod je určen, krom identifikátoru, dvojicí souřadnic x a y.

- *Linie* - Je definována svými dvěma krajními body. Její tvar je pak tvořen množinou geometrických bodů, které mají pouze souřadnice, nikoli však topologický význam. Tyto „vnitřní“ body rozdělují linii na lomenou čáru, proto pokud se má jednat o křivku, je zapotřebí většího počtu dělících bodů. V případě úsečky pak plně postačují pouze hraniční topologické body.
- *Polygon* - Definován je hraniční křivkou, která má počáteční i koncový bod shodný a bodem umístěným kdekoliv uvnitř plochy ohraničené linií.



Obrázek 2.6: Znázornění elementárních prvků Bod, Linie, Polygon

V některých geografických informačních systémech jsou body reprezentovány jako speciální případ linie, kdy první a poslední bod jsou shodné a mezi nimi neleží žádný další geometrický bod [12].

2.5 Topologie ve vektorových vestvách

Jedním z hlavních rozdílů mezi vektorovými a rastrovými daty je informace o topologii objektů, explicitní vyjádření o jejich vzájemné poloze. Příkladem pro topologickou relaci v geografických systémech může být informace o sousedství dvou států, pokud mají společnou hranici.

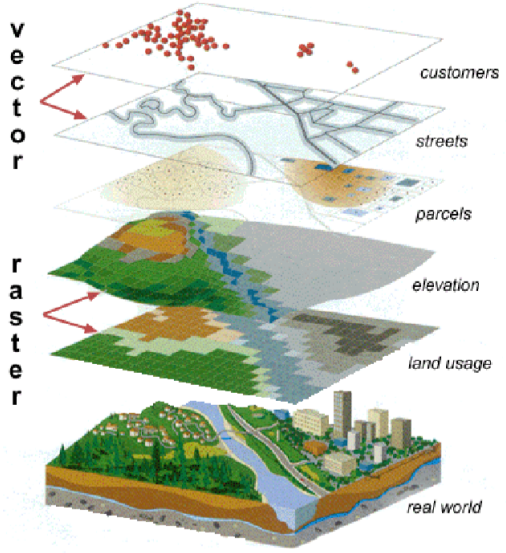
Mnoho algoritmů z vektorové analýzy lze provádět i bez znalosti topologie. Nicméně řada algoritmů tuto informaci vyžaduje a v tom se liší od rastrových formátů. Nejčastější aplikace topologie je v síťových analýzách, například výpočet nejkratší cesty, kdy je třeba znát sousední uzly, či rozdělení sítě do obslužných zón.

V geografickém systému GRASS je topologie dvojího typu. Takzvaná *pseudo - topologie*, která čítá výše zmíněné informace o sousedství, vlastnictví atd. a topologie *úplná*, která vyžaduje sloučení shodných linií, jako například hranic sousedících polygonů. Krom informace o sousednosti je jejich společná hranice pouze jeden objekt, nikoliv dva, pro každý polygon zvlášť.

Způsob uložení informace o topologie se liší GIS od GISu. Některé systémy tento údaj ukládají do zvláštního souboru s topologií, některé vytvoří záznam v souboru atributů, ...

2.6 Skládání vektorových vestev

Typickým přístupem GIS k datům je rozdělení informací do jednotlivých vrstev, které spolu tématicky souvisí. Příkladem může být vrstva vodstva, kde jsou zaznačeny veškeré vodní toky na mapovaném území, informace o směru jejich toku, jejich průměrné výšce hladiny a třeba okrsek, pod který daný úsek spadá. Na druhou stranu může existovat vrstva modelující nadmořskou výšku, což je například množina vektorů udávající velikosti gradientů a jejich hustotu.



Obrázek 2.7: Znázornění skládání modelu z jednotlivých vrstev [11]

Modelovaný systém je pak složení veškerých dostupných vrstev vztažených k tomuto systému. Složení jednotlivých vrstev do výsledného mapsetu je znázorněno na obrázku 2.7. Obrovskou výhodou této dekompozice je rozdělení problému na menší části a možnost zobrazování či počítání s pouze relevantními zdroji. Pokud je však potřeba počítat průnik tří vrstev, musí se postupně volat analytická operace pro dvě vrstvy, dokud nedosáhneme ekvivalentního výsledku.

V závislosti na používaném formátu uložení vektorových dat (viz 2.3) se liší fyzická reprezentace jednotlivých vrstev. Ty mohou být uloženy buď jako samostatné soubory (což je nejčastější případ), nebo odděleně v různých adresářích. Cílem pro knihovnu La'GIS je mít jeden soubor na jednu vrstvu.

Obecně lze říci, že slučování vrstev je jednoduchá operace, jejich rozdělování však nikoliv. Proto se vrstvy ukládají odděleně, aby byla možná jejich přenositelnost a zapojení tak do jiného datového kontextu.

2.7 Současná technologie

Mnoho současných oborů se zabývá vývojem geografických systémů. Některé vznikly jako akademické systémy a poté přešly do rukou veřejnosti, jiné slouží jako komerční produkty konkrétních firem. Jak se jednotlivé GIS liší historií, tak se liší i přístupem k uživateli. Existují jak konzolové GIS, tak s grafickým rozhraním. Zde bude uveden stručný výčet konkrétních systémů a jejich charakteristika.

Program GRASS

Geografický systém GRASS (v originálním znění *Geographic Resources Analysis Support System*) je systém určený pro organizaci prostorových dat, jejich zpracování, grafickou reprezentaci a vizualizaci. Původně byl tento projekt vyvíjen armádním sektorem U.S. Army. V současnosti je používán jak akademickým prostředím, tak i komerční sférou [6].

GRASS je konzolová aplikace a veškeré vstupy mu tak mohou být dodány pomocí klávesnice. V případě potřeby však disponuje taktéž grafickým uživatelským rozhraním. Pro zobrazení dat může být vyvoláno grafické okno s náhledem na mapové vrstvy a jejich modifikace.

GRASS umí zpracovávat data ve vektorovém formátu, v rastrovém (2D, 3D) či tyto formáty mezi sebou převádět (byť vznikají jisté nepřesnosti). Jedná se o veřejností široce rozvíjený software, proto jeho funkční sada neustále roste 2.8.

Tento systém sloužil jako hlavní zdroj informací a konceptů pro tuto diplomovou práci.

Obrázek 2.8: Ukázka posledních časů commitů vyvíjené verze GRASS.

Program ArcGIS

Systém ArcGIS je příkladem komerčně vyvíjeného geografického informačního systému. ArcGIS je obsáhlý nástroj pro organizaci, správu a distribuci prostorových dat. Tento systém je užíván ve všech možných odvětvích a profesích. Od vládních organizací, přes školní a výzkumné skupiny až po média.

Společnost, zabývající se vývojem tohoto systému, poskytuje též soubory vstupních dat pro využití softwarem ArcGIS. Dle oficiálních stránek společnosti lze kompilovat zdrojová data z různých zdrojů, čítaje data z geografických databází, souborů, fotografií a videí z konkrétních geografických poloh, satelitních snímků a mnoho dalšího [3].

Samotný software ArcGIS je desktopová aplikace se zaměřením na grafické uživatelské rozhraní. Poskytuje celou řadu analytických operací včetně 3D analýz.

Nástroj Oracle Spatial

Byť se v tomto případě nejedná o celý GIS, poskytuje tento nástroj od formy Oracle množství analytických operací nad (post)relačními databázemi. Mnoho GIS tak může využít již implementovaných analytických operací a datového skladu zároveň v podobě Oracle Spatial (celým název software Oracle Spatial and Graph).

Běžně implementovanými funkcemi tohoto produktu jsou operace jako měření vzdálenosti dvou objektů, počítání obsahu či obvodu polygonů, efektivní indexování dat, množinové operace a další.

Program OpenJump

OpenJump je otevřený geografický informační systém implementovaný v programovacím jazyce Java. Byť se jedná o ukázku dalšího open source GIS, není tak mohutně podporován a vyvíjen jako GRASS. O jeho vývoj se stará hrstka dobrovolníků z celého světa [4].

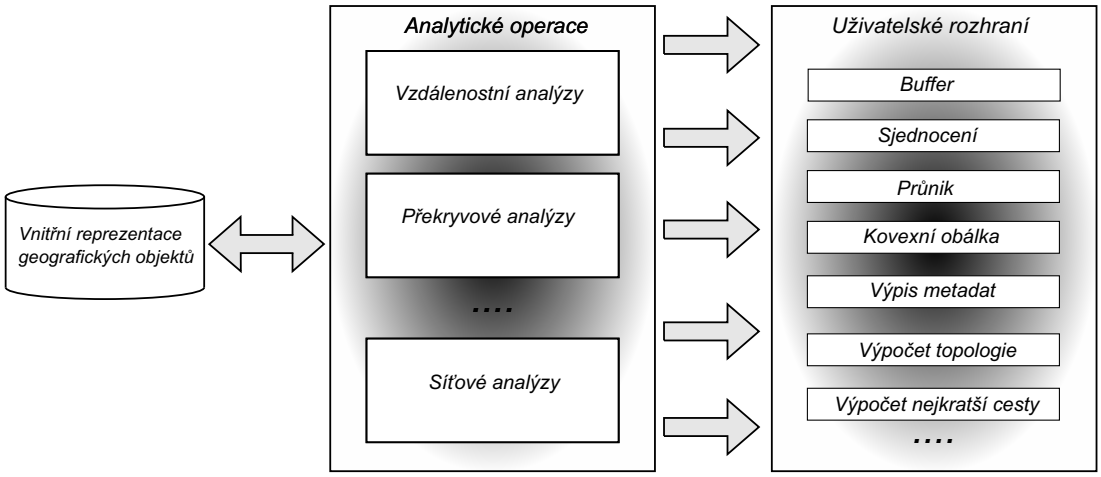
Program QGIS

QGIS je posledním zmíněným open source systémem. Jeho vznik je datován ke květnu roku 2002. Funguje na programovacím toolkitu Qt a jazyku C++. QGIS používá grafické uživatelské prostředí, které se snaží být co nejvíce uživatelsky přívětivé s původní myšlenkou poskytovat prohlížeč geografických dat. V současnosti jeho popularita stoupá a mnoho uživatelů využívá denně služeb QGIS [5].

Kapitola 3

Koncepty knihovny La'GIS

Tato kapitola popisuje návrh vnitřního formátu knihovny La'GIS. Abstraktní třídy a struktury jsou zvoleny tak, aby co nejintuitivněji popisovaly význam jednotlivých části geografického systémů. Přehled systému na nejvyšší zvolené úrovni abstrakce je možno vidět na obrázku 3.1.



Obrázek 3.1: Koncept geografické knihovny La'GIS

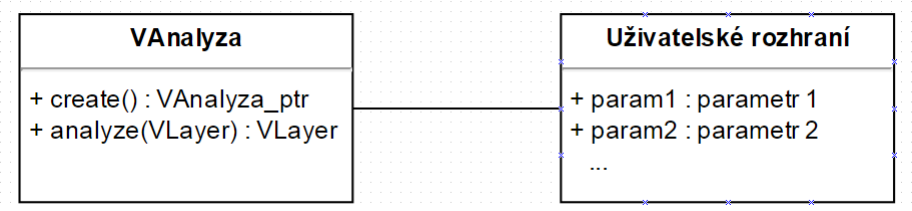
Knihovna je koncipována tak, aby se daly volat jednotlivé analytické operace a výstupem byla nová vrstva. Například nástroji pro analýzu překryvu se jako parametr předají dvě vrstvy a navrátí se vypočítaná vrstva.

3.1 Rozhraní knihovny La'GIS

Používání této knihovny se dá rozdělit do dvou kategorií:

1. Uživatelské - spouštění analýz pomocí binárních souborů.
2. Programátorské - využívání tříd modulů a jejich metod.

Aplikace je rozdělena na soubor jednotlivých modulů, které provádějí analytické operace. Současně jsou nad těmito moduly vystavěny malé spustitelné programy, které nad nimi zajišťují obsluhu. Rozdíl mezi těmito přístupy je patrný z obrázku 3.2.



Obrázek 3.2: Uživatelské a programátorské rozhraní knihovny

Příkladem spuštění vzdálenostní analýzy `buffer` přes uživatelské rozhraní je:
`./buffer -i tmp/london.db -d 100 -o tmp/london_b.db -s`

3.2 Vztahy abstraktních tříd

V této sekci je popsáno, jak jsou navrženy jednotlivé abstraktní třídy a jaké je jejich hierarchie. Nejedná se o kompletní popis implementace programu, zmíněny jsou jen třídy důležité z hlediska stavby aplikace.

3.2.1 Abstraktní třída VLayer

Třída `VLayer` implementuje vnitřní reprezentaci vektorové vrstvy. Součástí této třídy je seznam geografických objektů. Instance třídy `VLayer` si uchovává informace o metadatech vrstvy a je přímo napojena na konkrétní soubor s geografickými daty.

Součástí atributů třídy je i kompletní informace o metadatech, které jsou popsány v kapitole 3.2.2. Tyto metadata si objekt vrstvy vypočítá sám.

Tento objekt slouží pro primární komunikaci s geografickými daty. Vyhledání geografických objektů ve vrstvě se provádí na základě zadání katalogového čísla. Přes toto číslo lze získat buď přístup přímo k danému objektu, nebo se na něj pouze odkazovat v rámci určité operace.

Samotná vrstva neprovádí žádné analytické operace či výpočty s daty. Třída však má však přiřazenou zodpovědnost za konzistenci vektorových dat a jejich ukládání či načítání z disku.

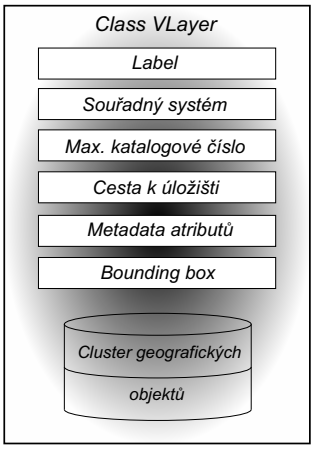
Na obrázku 3.3 je vidět vztah mezi abstraktní třídou vrstvy a množinou vektorových geografických dat.

3.2.2 Metadata vrstvy

Jelikož třída `VLayer` zastupuje v subsystému knihovny La'GIS tématickou vrstvu vektorových objektů, je třeba jí nastavit nějaké vlastnosti, které ji zasadí do většího kontextu. Z pohledu uživatele je vhodné, aby mohla mít vrstva přiřazen nějaký název (label). Pro práci s atributy dat je ve třídě `VLayer` uložen seznam všech atributů, respektive jejich název a datový typ.

Z hlediska geografie je nutné uchovávat informaci o použitém náhradním elipsoidu, případně o typu provedené projekce.

Aby se některé informace nemusely vypočítávat neustále znovu, je z hlediska optimalizace v metadatech vrstvy uloženo nejvyšší katalogové číslo a rozměry a souřadnice bounding boxu vrstvy.



Obrázek 3.3: Abstrakce třídy VLayer

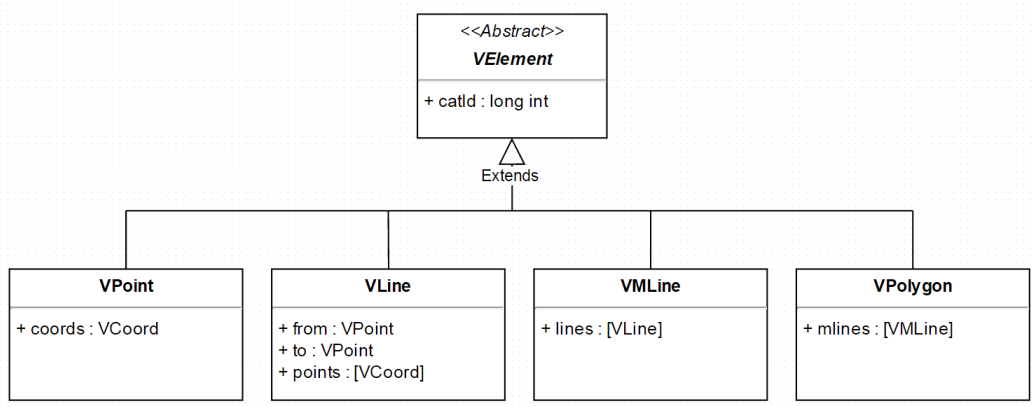
3.2.3 Třídy geografických objektů

Při pohledu na geografický objekt s vysokou mírou abstrakce, je každý modelovaný objekt spjat s objektem na povrchu Země. Z tohoto hlediska také vychází návrh nejvyšší abstraktní třídy `VElement`, která poskytuje rozhraní pro konkrétní typy primitiv.

Abstraktní třída VElement

Tato třída reprezentuje obecný geografický objekt a poskytuje základní rozhraní pro manipulaci s ním. Na obrázku 3.4 je znázorněn vztah mezi touto abstraktní třídou a třídami z ní odvozenými.

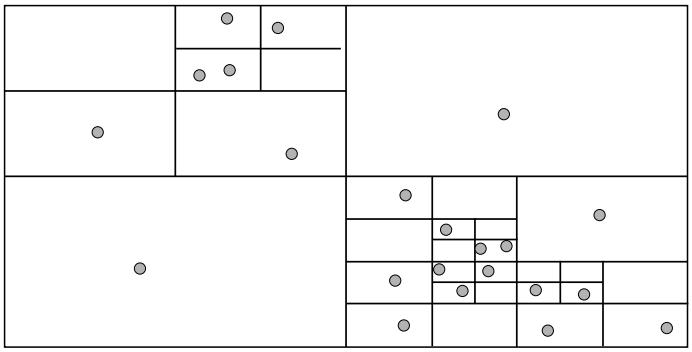
O třídách `VPoint`, `VLine`, `VMLine` a `VPolygon` detailněji pojednává kapitola 4.



Obrázek 3.4: Znázornění vztahu mezi abstraktní třídou `VElement` a třídami specializovaných tříd `VPoint`, `VLine`, `VMLine` a `VPolygon`

3.2.4 Abstraktní třídy atributů geografických objektů

Výhoda GIS spočívá v možnosti ukládat dodatečné informace ke geografickým datům. To je prováděno formou atributů. Každý atribut je složen z jména, které musí být v rámci vrstvy



Obrázek 3.5: Příklad rozložení prostorového indexu Quadtree

unikátní a hodnoty. Knihovna La'GIS podporuje atributy datového typu `INT` a `FLOAT`. Dalšími případnými datovými typy, které by mohla knihovna případně podporovat jsou `TEXT` nebo `DATE`. O způsobu uložení atributů objektů pojednává kapitola 4.

3.2.5 Abstraktní třída prostorového indexu

Protože v datasetu vektorové vrstvy může být velké množství geografických objektů, je vhodné nad nimi vybudovat indexační strukturu. Pro řadu vektorových analýz se pak značně zmenší výpočetní prostor. Pro užití prostorového indexu byly uvažovány dva přístupy:

1. Rovnoměrný, dlaždicový index
2. Indexační struktura typu *Quadtree*

Dlaždicový index by byl výhodný v tom, že se snadno zjišťují okolní buňky indexu. Jeho nevýhoda vychází z rovnoměrného rozložení po celé ploše vektorové vrstvy, čili některé buňky budou prázdné, některé přesyceny daty. Dalším problémem je určení správné velikosti buňky.

Na základě těchto důvodů byla zvolena indexační struktura typu *Quadtree*. Tento typ indexu se dynamicky přizpůsobí hustotě vektorových dat. Zároveň žádná vektorová analýza knihovny La'GIS nepotřebuje pro svůj výpočet zjistit okolní buňky indexu. Jedinou nevýhodou užití tohoto přístupu v této knihovně je stromový přístup ke koncovým buňkám struktury. V případě dlaždicového indexu by byl přístup konstantní, v případě *Quadtree* je přístup logaritmický. Na obrázku 3.5 je znázorněna možná struktura prostorového indexu.

3.2.6 Abstraktní třída úložiště

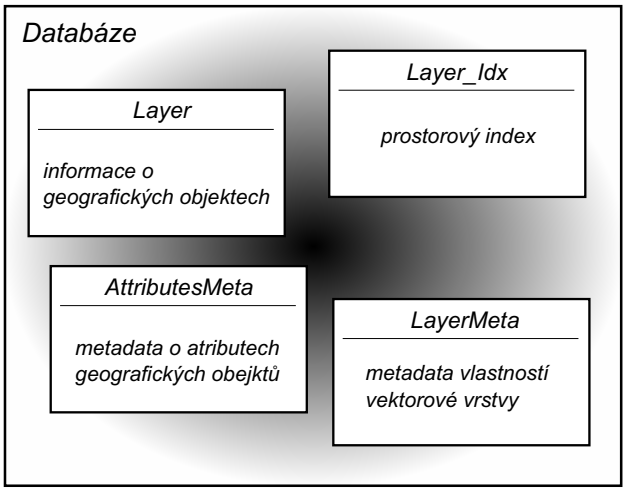
Jak již bylo zmíněno výše, cílem knihovny je ukládat veškerá data o vrstvě do jednoho souboru. Třída `Storage` poskytuje rozhraní pro přístup k uloženým vektorovým datům na disku. Tato abstrakce zajišťuje užití více typů úložiště. Implicitně je nastaveno ukládání ve formě databáze.

Třída je navržena pro přímou komunikaci s třídou `VLayer`, případně s konkrétními geografickými objekty. To znamená, že čtení či zápis inicializuje vektorová vrstva.

Třída Database

Knihovna La'GIS je implementována s podporou pouze jednoho způsobu ukládání dat. To zajišťuje třída `Database`. Třída využívá funkcí otevřené knihovny `sqlite3`.

Databáze obsahuje tabulky s vektorovou informací o objektech, záznam o metadatech, tabulku prostorového indexu a podobně. Koncept databáze lze vidět na obrázku 3.6. Konkrétní formát ukládaných dat a struktura databáze je popsána v kapitole 4.



Obrázek 3.6: Koncept databáze pro ukládání dat o vektorové vrstvě

3.2.7 Třídy vektorové analýzy

Vektorové analýzy jsou základní funkcionalitou GIS. V této sekci jsou popsány abstraktní třídy pro jednotlivé analýzy.

Základní myšlenka výpočtu analytických operací je předání vektorové vrstvy (případně dvou vrstev) nějaké třídě, která vypočítá vrstvu novou. Důvodem pro zvolení tohoto konceptu je co největší zapouzdření operací a abstrakce operací. Na obrázku 3.7 lze vidět typické analytické operace mezi rozdílnými typy geografických dat.

Analytické operace můžeme rozdělit do čtyř skupin v závislosti na tom, jakým typem výpočtu se zabývají.

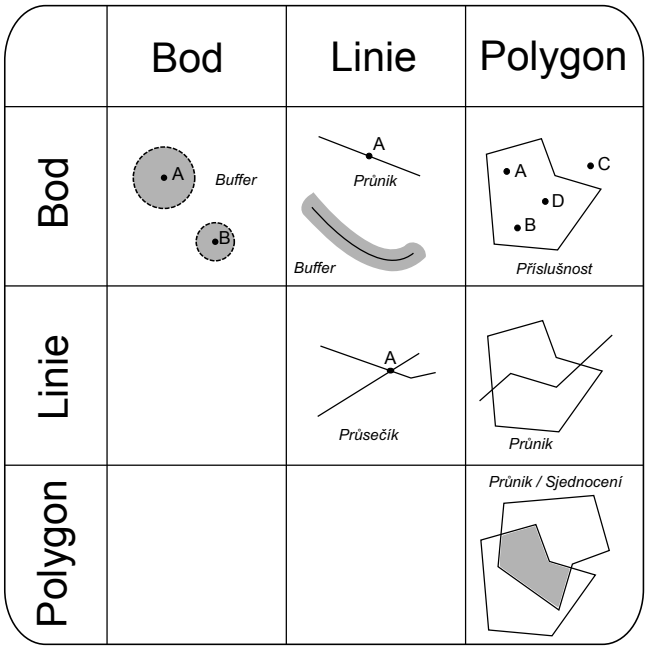
Třídy pro vzdálenostní analýzy

Vzdálenostní analýzy jsou analýzy, při kterých se řeší nějaký prostorový problém a je nutno se zabývat vzdálenostmi. Příkladem může být operace *buffer* (někdy též *offset*) nebo selekce podle vzdálenosti (viz selekční analýzy 14).

Operaci *buffer* implementuje třída `VBufferBuilder`. Vstupem pro tuto třídu je jedna vrstva a vzdálenost, pro kterou je analýza provedena. Výstupem je vrstva, která obsahuje množinu geografických objektů typu `VPolygon`.

Při výpočtu této analýzy nad celou vrstvou se využívá funkcí dalšího modulu (`VOverlay`), o tom více v kapitole 4.

Uvedený algoritmus 1 je platný pouze pro polygony. Pro ostatní typy primitiv je třeba jeho modifikace. O potřebných modifikacích a jejich implementaci pojednává kapitola 4. Bez korekce polygonu by uvedený algoritmus měl časovou složitost $O(n)$. Protože však mohou výpočtem vznikat tzv. nejednoduché polygony (mají smyčky) a ty jsou v geografických systémech nelegální, je třeba provést určitou korekci. Princip korekce je popsán v kapitole 4.



Obrázek 3.7: Příklady typických operací mezi jednotlivými primitivy [14].

Algoritmus 1: Pseudokód algoritmu pro vzdálenostní analýzu buffer

Input: Polygon `P`
Input: Vzdálenost pro výpočet bufferu `D`
Output: Vypočítaný polygon `R`

```
if D = 0 then return P;
if orientace P = clockwise then side := LEFT;
else side := RIGHT;
coords := posloupnost souřadnic hraniční linie P;
vects := posloupnost vektorů tvořící hraniční linii P;
coordsR := empty;
foreach v : vects do
    norm := normála vektoru v na stranu side;
    shift := posun části hraniční linie o D ve směru norm;
    vložení shift do coordsR;
end
R := vytvoření polygonu z posloupnosti coordsR;
korekce polygonu R;
return R;
```

Třída pro selekční analýzy

Selekčními analýzami rozumíme operace, které na základě nějakého predikátu vyberou pouze ty geografické objekty, které ho splňují. Tuto funkcionalitu zajišťuje třída `FetchRequest`.

Predikáty mohou být následujících typů:

- atributový
- na ohraničení (bounding box)
- vzdálenostní
- na katalogové číslo
- na typ geografického objektu

Pro konstrukci atributového a vzdálenostního predikátu je implementována podpora porovnávacích operátorů (<, >, =atd.). Ostatní predikáty se testují na shodu, případně na příslušnost do zvolené oblasti.

Mezi všemi nastavenými predikáty je logický operátor `AND`, čili pro výsledek selekce je nutno splnit všechny uvedené podmínky.

Jelikož se počítá pouze s operátorem `AND`, je pro následující algoritmus 2 významné pořadí vyhodnocování podmínek. Více k této optimalizaci je napsáno v kapitole 4.

Algoritmus 2: Pseudokód algoritmu pro selekční analýzu

Input: Množina geografických objektů `I`
Input: Seznam predikátů `P`
Output: Množina geografických objektů splňující `P`

```
1 items := I;
2 foreach p : P do
3 |   passing := empty;
4 |   foreach item : items do
5 |   |   if item splňuje predikát p then vlož item do passing;
6 |   end
7 |   items := passing;
8 end
9 return items;
```

Třída pro překryvové analýzy

Z obrázku 3.7 lze vyčíst, že mezi časté operace nad polygony patří takzvané překryvové analýzy. Typickými množinovými operacemi této analýzy jsou *sjednocení*, *průnik*, *doplněk* a *rozdíl*. Tuto analýzu implementuje třída `VOverlay`.

Knihovna La'GIS z výše zmíněných operátorů podporuje pouze sjednocení a průnik. Algoritmus 3 popisuje princip operace průniku. Operaci sjednocení popisuje algoritmus 4. Tyto algoritmy popisují provedení operace pouze nad samostatnými geografickými objekty. Pro výpočet analýzy nad celou vrstvou jsou tyto algoritmy obaleny dalším algoritmem. Ten je také popsán v kapitole 4.

Algoritmus průniku nepracuje přesně v jednom (známém) konkrétním případě. A to, pokud by výsledek operace měly být dva geografické objekty. V tomto případě se vypočítá pouze jeden z nich.

Algoritmus 3: Pseudokód algoritmu pro operátor průniku

Input: Polygon `A`
Input: Polygon `B`
Output: Průnik polygonů `C`

```
1  if orientace A != clockwise then otoč orientaci A;
2  if orientace B != clockwise then otoč orientaci B;
3  najdi průsečíky A a B;
4  ulož je na správné místo v hraniční linii polygonu A;
5  ulož je na správné místo v hraniční linii polygonu B;
6  minXY := najdi bod, od kterého jsou oba polygony v 1. kvadrantu;
7  init := bod A nebo B, který je nejblíže minXY;
8  start := nejbližší průsečík v clockwise směru od bodu init;
9  act := start;
10 coordsC := empty;
11 repeat
12 |   vlož act do coordsC;
13 |   if act = průsečík then
14 |   |   změň polygon, po kterém se iteruje
15 |   end
16 |   act := další bod v posloupnosti iterovaného polygonu v counter-clockwise
   |   orientaci;
17 until start = act;
18 C := vytvoř polygon z posloupnosti coordsC;
19 return C;
```

Algoritmus 4: Pseudokód algoritmu pro operátor sjednocení

Input: Polygon `A`
Input: Polygon `B`
Output: Sjednocení polygonů `C`

```
1  if orientace A != clockwise then otoč orientaci A;
2  if orientace B != clockwise then otoč orientaci B;
3  najdi průsečíky A a B;
4  ulož je na správné místo v hraniční linii polygonu A;
5  ulož je na správné místo v hraniční linii polygonu B;
6  minXY := najdi bod, od kterého jsou oba polygony v 1. kvadrantu;
7  init := bod A nebo B, který je nejblíže minXY;
8  act := start;
9  coordsC := empty;
10 repeat
11 |  vlož act do coordsC;
12 |  if act = průsečík then
13 |  |  změň polygon, po kterém se iteruje
14 |  end
15 |  act := další bod v posloupnosti iterovaného polygonu;
16 until start = act;
17 C := vytvoř polygon z posloupnosti coordsC;
18 return C;
```

Třída pro síťové analýzy

Síťové analýzy jsou výpočty, které se provádí nad grafem (množinou uzlů a hran). V případě GIS se pak jedná o operace nad vrstvou linií a vektorových bodů, které dohromady tvoří síť.

Tyto analýzy počítá třída `Net`. Knihovna La'GIS implementuje tři operace z této rodiny. Jedná se o výpočet nejkratší cesty, rozdělení sítě k mezi centra a rozdělení sítě do obslužných zón.

Základním operace, ze které vychází i zbylé dvě, je výpočet nejkratší cesty. Tato operace je implementována algoritmem `A*`, který je popsán pseudokódem 5. Analýzy rozdělení zdrojů a obslužných zón jsou popsána algoritmy 6 a 7.

Algoritmus 5: Pseudokód algoritmu A*

Input: Síť `N`
Input: Startovní bod `start`
Input: Cílový bod `goal`
Output: Posloupnost objektů, přes které vede cesta `P`

```
1  closed := empty;
2  open := empty;
3  came_from := empty;
4  g_score[start] := 0;
5  f_score[start] := heuristika(start, goal);
6  while open is not empty do
7      current := uzel z open s nejnižším f_score;
8      if current = goal then return rekonstruuj cestu z came_from;
9      odeber current z open;
10     přidej current do closed;
11     foreach neighbor : sousední uzly current do
12         if neighbor je v closed then pokračuj na další neighbor;
13         tg_score := g_score[current] + vzdálenost mezi current a neighbor;
14         if neighbor není v open nebo tg_score ¡ g_score[neighbor] then
15             came_from[neighbor] := current;
16             g_score[neighbor] := tg_score
17         end
18         ;
19         f_score[neighbor] := g_score[neighbor] + heuristika(neighbor, goal);
20         if neighbor není v open then přidej neighbor do open;
21     end
22     return neúspěch;
23 end
```

Algoritmus 6: Pseudokód algoritmu pro výpočet rozdělení zdrojů

```
Input: Síť N
Input: Centra C
Output: Vypočítaná síť NN
inicializuj vzdálenost d0 všech uzlů sítě N na max;
foreach centrum : C do
    foreach node : N do
        d := vzdálenost z node do centrum;
        if d < d0 then
            d0[node] := d;
            closest[node] := centrum;
        end
    end
end
NN := vytvoř síť z N s přidanými atributy closest;
return NN;
```

Algoritmus 7: Pseudokód algoritmu pro výpočet obslužných zón

```
Input: Síť N
Input: Centra C
Output: Vypočítaná síť NN
inicializuj vzdálenost d0 všech uzlů sítě N na max;
foreach centrum : C do
    foreach node : N do
        d := vzdálenost z node do centrum;
        if d < d0 then
            d0[node] := d;
        end
    end
end
foreach node : N do
    zone[node] := přiřazení obslužné zóny na základě d0[node];
end
NN := vytvoř síť z N s přidanými atributy zone;
return NN;
```

Kapitola 4

Implementace knihovny La'GIS

Tato kapitola popisuje technické provedení knihovny La'GIS. Jsou zde popsány principy řešení jednotlivých implementačních problémů. Knihovna je implementována v jazyce C++, proto zde uvedené příklady a názvy objektů jsou z množiny tohoto jazyka.

4.1 Implementace třídy VLayer

Tato třída implementuje kompletní vektorovou vrstvu. To znamená, že krom samotných geografických dat jsou v ní uloženy metadata vrstvy, souřadnicový systém, prostorový index nad daty, přístup k fyzickému uložení a podobně.

Protože za správu geografických dat odpovídá instance třídy `VLayer`, musí tato třída implementovat metody pro vytváření nových objektů. Toho se využívá prakticky při všech analytických operacích, při kterých se vytváří nová vrstva. Tyto operace zajišťují metody `VLayer::createPoint()`, `VLayer::createLine()`,...

Protože knihovna je navržena tak, aby se daly provádět analytické operace těsně za sebou, může se během úprav dostat do „nekonzistentního" stavu. To znamená, že například nebude platný prostorový index (viz 4.4. Proto je nutné po ukončení všech výpočtů invokovat metodu `VLayer::finalize()`, která tyto vlastnosti znovu přepočítá. V případě uživatelského rozhraní se tak děje na konci každé spustitelné analýzy (pokud je požadováno uložení vrstvy).

4.1.1 Kontejner pro geografická data

Třída `VLayer` uchovává geografická data ve struktuře `cluster_t`, což je asociativní pole. Klíčem v tomto poli je katalogové číslo geografického objektu, hodnotou pak ukazatel na objekt. Tato struktura byla zvolena na základě předpokladu, že posloupnost katalogových čísel nemusí být spojitá. V případě řídkého pole katalogových čísel se pak velikost kontejneru přizpůsobí počtu objektů. Tato implementace sebou nese i nevýhodu, kterou je logaritmická časová složitost při vyhledávání katalogového čísla. Za všechny geografické objekty nese třída `VLayer` plnou zodpovědnost, což je implementováno uložením silného ukazatele na objekt:

```
typedef std::map<catId_t, VElement_ptr> cluster_t
```

4.1.2 Množina finálních geografických objektů

Jelikož složitější geografická data (od linie výš) jsou složeny z menších částí, obsahuje kontejner třídy `VLayer` velké množství dat. Přitom většina z nich není pro výpočet vektorové analýzy potřeba. Proto je implementován další kontejner. Případná vektorová analýza si pak načte pouze objekty tohoto seznamu. V tomto kontejneru jsou drženy slabé ukazatele na geografické objekty, které jsou tzv. finální (nesoucí geografický význam). Tento kontejner implementován formou asociativního pole:

```
typedef std::map<catId_t, VElement_wptr> cluster_wt
```

4.1.3 Odložené načítání

Co se týče načítání vektorové vrstvy z disku, jsou implementovány dva přístupy. První, kompletní načítání, načte celý seznam geografických objektů, jejich atributů, geometrie a podobně. V řadě analytických operací však nejsou kompletní data potřeba. Druhý přístup je tzv. odložené načítání. To znamená, že při vytváření instance třídy `VLayer` se načte pouze seznam dvojic katalogové číslo - typ objektu. Pokud jsou pak potřeba nějaká data, která nejsou načtena, provede se dočtení všech hodnot pro ten jeden konkrétní objekt. Cílem této implementace je co nejvíce snížit množství dat načítaných z disku.

4.1.4 Metadata vektorové vrstvy

Metadata vrstvy se dají rozdělit na dvě skupiny, podle toho, kdy vznikají:

1. Definovaná - Metadata jsou zadávaná uživatelem.
2. Vypočítaná - Aplikace si sama dopočítá potřebná data.

Mezi definovaná metadata patří název vrstvy (label) a informace o geografickém souřadném systému. Mezi vypočítaná nejvyšší katalogové číslo, bounding box vrstvy a atributy. Další možná rozšíření jsou počty jednotlivých typů geografických objektů nebo informace o provedení výpočtu topologie. Tyto informace však nebyly potřebné k žádné analytické operaci, proto není jejich podpora implementována.

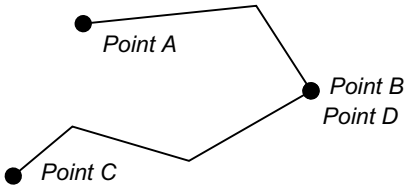
Metadata o atributech vektorových dat obsahují informace o názvu atributu a jeho datový typ. Pro přidání nového druhu atributu geografickým objektům je nutno „zaregistrovat“ tento atribut v instanci třídy `VLayer`. To se provede voláním metody `VLayer::addAttributeMeta(std::string label, type_t dataType)`. Datový typ atributu je nutno zadat kvůli zvolení správné třídy atributu a kvůli datovému typu v databázi.

Způsob uložení metadat na disk je popsán v sekci 4.5.

4.2 Implementace geografických objektů

Geografický objekt je reprezentován třídou `VElement`. Jedná se o abstraktní třídu pro všechny typy geografických objektů. Tato třída poskytuje společné rozhraní pro konkrétní typy objektů, které z ní dědí.

Mezi základní operace, kterých je často využíváno vektorovými analýzami, patří vyjádření geografického objektu formou posloupnosti souřadnic, vytvoření vlastní kopie nebo porovnání na shodu s jiným geografickým objektem.



Obrázek 4.1: Případ, kdy na sebe navazující linie A - B a C - D nemají žádný společný bod

V sekci 2.4 jsou popsána tři běžně užívané typy geografických objektů. Já jsem si tuto množinu rozšířil o čtvrtý typ. Knihovna La'GIS tak pracuje s těmito geografickými typy objektů:

- Bod - třída `VPoint`
- Linie - třída `VLine`
- Multilinie - třída `VMLine`
- Polygon - třída `VPolygon`

4.2.1 Implementace geografického bodu

Specifikum třídy `VPoint` je, že se neskládá z žádného dalšího geografického primitiva. Jedná se o základní stavební prvek všech ostatních topologických objektů. Součástí této třídy je informace o umístění ve 2D souřadném systému. To je implementováno třídou `VCoord`.

Z geografického hlediska není logické, aby dva topologické body měly stejné souřadnice. Sama tato třída však nijak nezajišťuje validaci, zda se na konkrétních souřadnicích nenachází jiný bod. Touto problematikou se zabývá analýza topologie, viz sekce 4.7.7.

4.2.2 Implementace geografické linie

Geografickou linii implementuje třída `VLine`. Z návrhu geografických primitiv vychází, že se skládá ze dvou topologických bodů a libovolného počtu geometrických bodů. Tímto způsobem se tvoří křivky (respektive lomené čáry).

Uložení krajních je implementováno formou slabého ukazatele na objekty typu `VPoint`, které jsou uloženy v instanci třídy `VLayer`. Geometrické body jsou reprezentovány kontejnerem `std::vector`, který obsahuje objekty třídy `VCoord`.

Napojení dvou linií na sebe musí být provedeno přes jeden z krajních bodů. Napojením dvou linií v místě, kde se protínají a není tam žádný z topologických bodů, řeší třída `VMLine`.

V případě, že nad vektorovou vrstvou nebyla provedena topologická analýza, mohou dvě na sebe navazující linie nemít společný žádný topologický bod. Tato situace je znázorněném na obrázku 4.1.

4.2.3 Implementace shluku geografických linií

Tuto třídu geografických objektů jsem zavedl kvůli větší robustnosti a logice struktury knihovny La'GIS. Implementovaná je pod názvem `VMLine`. Uplatnění je vysvětleno na následujícím příkladu.

Mějme dvě linie `A` a `B`, které se kříží. V místě křížení chceme vytvořit nový bod a ten přidat do obou linií. Z definice geografického objektu linie to ale není možné, neboť ta je definována pouze dvěma topologickými body.

Vytvoříme tedy pro každou linii „nadobjekt“, který bude obsahovat dvě linie vzniklé rozdělením původní linie průsečíkem.

Krom uvedeného příkladu lze multilinii užít i v případě, že se jedná o jednu linii, která z nějakého důvodu nemůže být spojitá.

Třída `VMLine` má informaci o obsažených liniích uloženou formou třídy `std::vector`, protože pro význam multilinie je důležité pořadí linií. Stejně jako v případě třídy `VLine` si i tato třída udržuje v kontejneru slabé ukazatele na linie.

4.2.4 Implementace geografické oblasti (polygonu)

Třída `VPolygon` se mírně liší od obdobných tříd používaných v jiných GIS. Tato třída obsahuje orientovaný seznam multilinií, které značí hranice oblasti. Z rekurzivní definice objektu pak plyne, že jeden geografický objekt třídy `VPolygon` je složen z minimálně jedné multilinie, minimálně jedné linie a jednoho geografického bodu. Tento bod taktéž značí místo, od kterého se provádějí výpočty s polygonem. Systém uložení jednotlivých multilinií je analogický s třídou `VMLine`.

Přístup skládání polygonu z multilinií má své opodstatnění ze stejného důvodu jako pro multilinie.

Současná verze knihovny La'GIS nepodporuje tzv. polygonu s dírami. To znamená polygony, které nemají svoji vnitřní plochu souvislou. Obvykle jsou tyto díry implementovány hraniční linií, která je orientovaná opačným směrem než hranice polygonu. Současná implementace stavby polygonu je vhodná pro rozšíření na polygon s dírami.

4.3 Atributy geografických dat

Instance třídy `VElement` implementuje podporu pro ukládání atributů. Implementovány jsou atributy typu `INT` a `FLOAT`. Společné rozhraní pro manipulaci s atributy poskytuje třída `VAttributeValue`, která se podle typu dat rozděluje na `VAttributeValueInt` a `VAttributeFloat`. Základními operacemi této třídy je načtení a uložení na disk a vrácení uložené hodnoty. Hodnota atributu je vrácena v textové podobě (objekt třídy `std::string`). Proto je potřeba ho explicitně konvertovat na cílený typ.

Některé analýzy si ukládají pomocné atributy, které lze později dále využít. Například při výpočtu topologie se ukládá každému geografickému objektu jeho délka, aby se počítala pouze jednou pro konkrétní vektorová data.

4.4 Implementace prostorového indexu

Prostorový index implementuje třída `QuadTree`. Jedné se o rekurzivní strukturu typu strom, ve které se každý uzel dělí do čtyř větví. Uzly struktury se rozlišují na *terminální* a *neterminální*, což je rozlišeno boolovskou hodnotou.

Prostorový index má uložený slabý ukazatel na každý prvek, který do něj, třeba i částečně, zasahuje. Bod může být logicky pouze v jedné buňce, ostatní typy primitiv i ve více. Této vlastnosti se využívá pro užití metody `QuadTree::neighbors()`, která vrátí seznam geografických objektů, které mají nějaký společný index s referenčním objektem. O implementaci této metody a jejím použití je odstavec 4.4.5.

4.4.1 Organizace vnitřní struktury indexu

V průběhu implementace byly vyzkoušeny dvě verze stromu. V první byly geografické objekty zaznačeny jen v terminálních uzlech. Ve druhé verzi jsou uloženy i v neterminálních. Výhodou při druhém přístupu je rychlejší vyhledávání prvku ve struktuře. Pokud neobsahuje rodičovský uzel hledaný objekt, nemá cenu prohledávat ve svých potomcích. Nevýhodou je větší spotřeba operační paměti, zejména při velkém objemu dat. Využití této implementace je zmíněno v sekci 4.4.5.

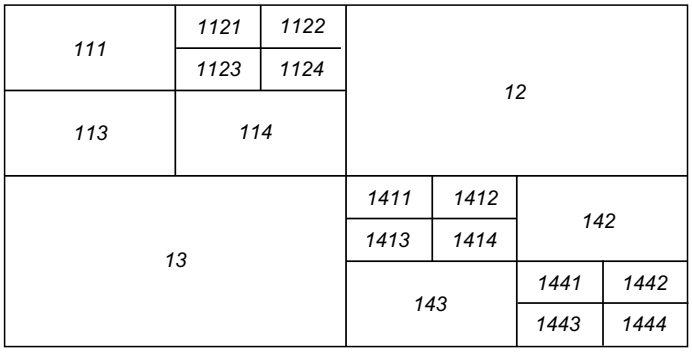
4.4.2 Omezení hloubky stromu a velikost buňky

Cílem prostorového indexu je snížit výpočetní prostor. Není proto třeba, aby buňka struktury obsahovala jen jeden, dva, či řádově jednotky objektů. Pro analytické operace stačí, pokud se jen výrazně redukuje množina všech geografických objektů. Experimentálně byla kapacita buňky nastavena na hodnotu 20 - tato hodnota je uložena v makru `QUADTREE_MAX_Q`.

Zároveň není žádoucí, aby se v případě velkého shluku objektů struktura mořila do nepřiměřené hloubky. I kvůli značení jednotlivých uzlů, není možné, aby byla hloubka stromu příliš velká. Vysvětlení je popsáno v sekci 4.4.3.

4.4.3 Značení uzlů QuadTree

Ve stromové struktuře prostorového indexu je nutné, aby byl každý uzel nějak identifikovatelný. V případě knihovny La'GIS je tento problém implementovaný formou celého čísla, které určuje cestu ke kořeni struktury. Toto značení je i intuitivní pro vývojáře, protože z něj lze snadno vyčíst polohu a hloubku uzlu. Příklad značení buněk je na obrázku 4.2. Použití tohoto způsobu značení je však omezením pro délku identifikátoru buňky. Jelikož je pro identifikátor použit datový typ `unsigned int`, jsme omezeni na délku desíti číslic (maximální hodnota `unsigned int` je 4 294 967 296). Hloubku zanoření stromu tak limituje i konstanta `QUADTREE_MAX_DEPTH`.



Obrázek 4.2: Znázornění způsobu identifikace buněk prostorového indexu

4.4.4 Přidávání geografických objektů do struktury

Třída `QuadTree` podporuje dvě varianty přidávání objektů do struktury. První je založena na přístupu „Rozděl a panuj“, kdy se kořenovému uzlu předá seznam všech objektů, jež je třeba uložit. Tato metoda je implementována funkcí `QuadTree::import()` a je mírně rychlejší, než inkrementální vkládání operací `QuadTree::insert()`. Inkrementální vkládání

je tou druhou možností. Tato varianta má své uplatnění v případě, že indexační struktura již existuje a je třeba do ní vložit nový prvek. Toho se využívá například při selekci dle bounding boxu.

4.4.5 Vyhledávání neighbor objektů

Nejspíše nejčastěji používanou operací prostorového indexu je zjištění objektů, které spadají do stejných buněk jako referenční objekt. Díky stromové struktuře třídy `QuadTree` a vlastnosti, že i neterminální uzel má uchovanou informaci o obsažených objektech, lze relevantní objekty najít s logaritmickou časovou složitostí.

4.4.6 Sestavení struktury podle načtených dat z databáze

Samotnému načítání informací z databáze se věnuje sekce 4.5. Při načítání struktury prostorového indexu je ale důležité pořadí, v jakém se data načítají. Algoritmus vytvářející strukturu počítá s tím, že jakmile dostane nějakou dvojici `(id buňky, seznam objektů)`, tak již ve struktuře existuje uzel, která je přímým rodičem vytvářeného uzlu. Ve struktuře indexu se pak rodič vyhledá a na příslušný ukazatel se vytvoří nová buňka.

Knihovna La'GIS s touto informací implicitně pracuje a nijak ji neověřuje. Pokud je tedy manuálně upraven obsah souboru s prostorovým indexem, program nemusí pracovat správně.

4.5 Ukládání dat na disk - databáze

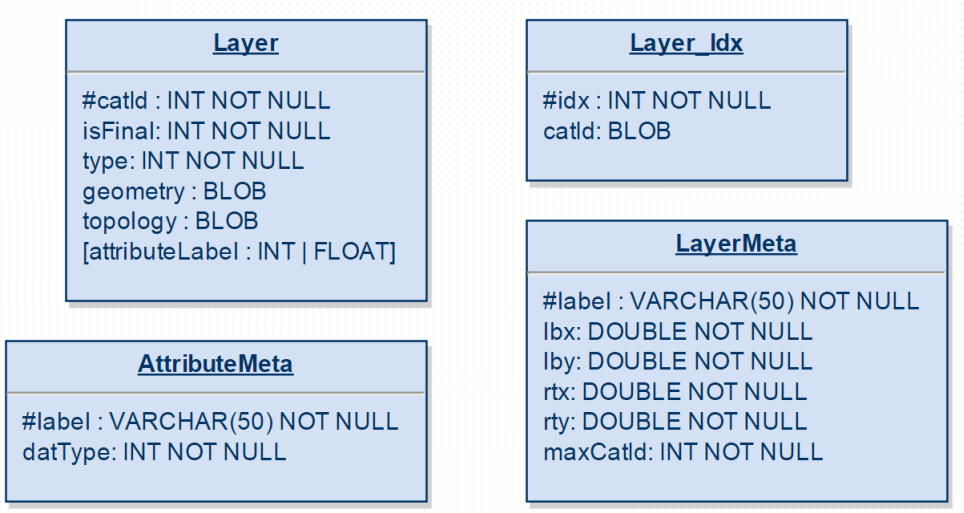
Tato část popisuje implementaci ukládání vektorových vrstev na disk a to formou databáze. Za tímto účelem je využívána otevřená knihovna `sqlite3`. V následujících odstavcích je popsáno schéma databázových tabulek, formátování dat, způsob komunikace s třídou `Database` a podobně. V této sekci je obecně popsáno veškerá funkčnost, která se pojí s databázovou komunikací knihovny La'GIS.

4.5.1 Schéma databáze vektorové vrstvy

Veškeré informace o vrstvě jsou uloženy v jediném souboru `sqlite3` databáze. Proto musí tato databáze obsahovat několik tabulek, které na sebe nutně nemusí mít žádnou vazbu. Hlavní tabulkou celé vrstvy je tabulka `Layer`. V té jsou uloženy všechny geografické objekty vrstvy, jejich geometrie, topologie, typy a podobně. Výhoda této implementace oproti geografickému systému GRASS je v tom, že pro atributy geografických dat je použita stejná tabulka. Všechny informace jsou tak na jednom místě a není třeba například při vyhledávání tvořit komplexní dotazy. Schéma celé databáze je popsáno diagramem na obrázku 4.3. Ostatní tabulky slouží pro uložení prostorového indexu, metadat vrstvy a metadata o atributech.

4.5.2 Ukládání dat do datového typu blob

Ze schématu databáze je patrné, že v určitých místech je použit datový typ `blob`. Tento datový typ je binární posloupnost nějakých dat. V následujících odstavcích je popsán význam jednotlivých užití a formát kódování.



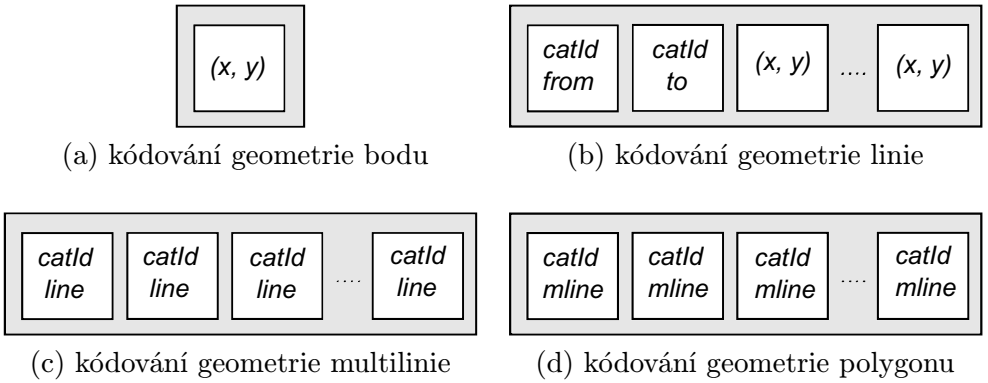
Obrázek 4.3: Schéma databáze knihovny La'GIS

Blob v tabulce Layer

Tabulka `Layer` obsahuje dva atributy, jejichž hodnota je kódovaná do binární posloupnosti. Jedná se o sloupce `geometry` a `topology`. Důvodem pro toto zakódování je rozdílná délka hodnot a formát kódování dat.

Atribut `geometry` popisuje geometrii geografického objektu. V závislosti na typu objektu se pak liší délka i formát uložených dat. Vše vychází z návrhu skládání jednotlivých primitiv z menších celků. Geografický objekt typu bod má v této struktuře zakódované pouze své x a y souřadnice. Linie má zakódovaná katalogová čísla na počáteční a koncový bod a následně posloupnost dvojic souřadnic pro vyjádření tvaru linie. Blob multilinie obsahuje orientovaný seznam katalogových čísel linií, obdobně polygon katalogová čísla multilinií. Formát kódování je patrný z obrázku 4.4.

Atribut `topology` vyjadřuje topologické vztahy s dalšími objekty. Tato hodnota může být prázdná, například v případě, že se jedná o samostatný bod. Formát kódování je pro všechny typy primitiv stejný. Blob obsahuje množinu katalogových čísel relevantních objektů. Datový typ `blob` je však použit kvůli rozdílné délce posloupnosti u každého objektu.



(a) kódování geometrie bodu

(b) kódování geometrie linie

(c) kódování geometrie multilinie

(d) kódování geometrie polygonu

Obrázek 4.4: Formáty kódování geometrie primitiv do binární posloupnosti

Blob v tabulce Layer_Idx

Tato tabulka obsahuje seznam identifikátorů indexů a k nim přiřazených binárních posloupností zakódovaných objektů. V tomto případě je `blob` pouze za sebe naskládaná katalogová čísla geografických objektů. Důvod, proč data kódovat a neukládat je přímo, je ten, že veškeré vyhledávání v indexační struktuře je prováděno v paměti RAM. Není třeba proto udržovat „čitelné“ informace v databázi, které by pouze navýšily velikost tabulky.

4.5.3 Ukládání atributů geografických objektů

Na rozdíl od jiných GIS je knihovna La'GIS implementovaná tak, aby byly geografické objekty a jejich atributy uloženy společně. Vyhledávání a načítání atributů je pak triviální. Komplikací ovšem je přidávání atributů do systému. Řešení je implementováno SQL operací `UPDATE`. Při vytvoření nového vrstvy (neboli metaatributu) se upraví tabulka `Layer`. Patřičná hodnota se pak uloží do nově vytvořeného sloupce. Sloupec s geografickým atributem je znázorněn na obrázku 4.3.

4.5.4 Ukládání dat do databáze

Komunikace s datovým skladem na disku zajišťuje pouze zmíněná třída `Database` za využívání funkcí knihovny `sqlite3`. Pokud nějaký objekt potřebuje svá data uložit na disk, musí mít dostupnou instanci této třídy. Samotná třída `Database` již očekává zakódovaná data v binární posloupnosti `blob`. Pro ukládání dat slouží například metody `Database::store()`, `Database::storeMeta()`, `Database::storeAttr()` a další.

4.5.5 Načítání dat z databáze

Třída `Database` implementuje metody `Database::load()`, `Database::loadMeta` a podobně pro načítání dat z disku do vnitřní reprezentace vektorových dat. Základní přístup, kdy se načítají kompletní data geografického objektu, je metoda `Database::load()`. Protože pro některé metody není třeba načítat všechny vlastnosti všech geografických objektů, byla implementována i metoda tzv. *odloženého načítání*.

Metoda odloženého načítání

Tato meto je implementována pro každé otevření vrstvy z disku. Celá vektorová vrstva si načte základní data, dvojice katalogové číslo a typ objektu, do operační paměti. Informace, že je objekt pouze částečně načten, je uložena formou příznaku `VElement::_loaded`. V případě, že jsou požadována data, která objekt nemá načtena (cokoliv, krom katalogového čísla nebo typu), je zavolána metoda `VLayer::load()`, která načte kompletní informaci o objektu. Příznak `VElement::_loaded` se při kompletním načtení nastaví na `true`.

4.5.6 Žurnálování databáze

Knihovna `sqlite3` využívá principy žurnálování. Tento přístup se může projevit značným omezením rychlosti zápisu na disk. Knihovna `sqlite3` typicky funguje tak, že po každém zapisovacím dotazu ověřuje stav databáze po zápisu - zda transakce proběhla v pořádku. Uzavřením celé zapisující posloupnosti do sekvence `begin` a `end` se kontrola provádí pouze na konci celé sekvence. Na vzorku padesáti geografických objektů se při použití `begin`, `end` urychlil zápis zhruba desetkrát.

4.6 Mapování objektů mezi vrstvami

V některých případech je nutné vytvořit kopii vybraných dat vektorové vrstvy. To může například nastat při skládání analytických operací za sebe. K tomuto účelu byly implementovány třídy `VLayerMapper`, `VClusterMapper` a `AttributeMapper`. Výsledkem mapování je nová vrstva, která obsahuje kopie geografických dat, atributů, metadat atd. referenční vrstvy.

4.7 Implementace vektorových analýz

V této sekci jsou popsány implementační problémy jednotlivých vektorových analýz.

4.7.1 Vzájemné využívání nezávislých modulů knihovny

Využívání modulů mezi vektorovými analýzami je jednou ze základních funkcionalit knihovny. Princip spočívá v tom, že se s vrstvou pracuje pouze v operační paměti. Je to tak jediný případ, kdy geografická data nejsou uložena trvale na disku. Takto vytvořenou vrstvu je pak možno převzít jako výstup analýzy A a použít ji pro vstup analýzy B. Tímto způsobem tak lze zřetězit libovolné množství výpočtů. Na závěr, po výpočtu všech operací, se nad vrstvou musí provést operace `VLayer::finalize()`. V některých následujících odstavcích jsou uvedeny příklady vzájemného využívání modulů.

4.7.2 Implementace modulu selekční analýzy

Princip výpočtu je popsán v kapitole konceptu knihovny 3. Z implementačního hlediska je důležitá část zabývající se pořadím vyhodnocování jednotlivých predikátů. Nejdříve je však třeba popsat, jakým způsobem se predikáty vytvářejí.

Třída `FetchRequest` implementuje operace pro přidávání jednotlivých typů podmínek k vyhodnocení. Příklady takových metod jsou `FetchRequest::addConditionDist`, reprezentující podmínku na vzdálenost, nebo `FetchRequest::addConditionType`, pro výběr dle typu geografického objektu. Každý typ predikátu je uložen do vlastního seznamu.

Před samotným provedením operace `FetchRequest::fetch` je třeba podmínky sloučit do jedné fronty predikátů. Podmínky se vyhodnocují v tom pořadí, v jakém jsou uloženy ve frontě. Sloužení predikátů implementuje metoda `FetchRequest::_appendConditions()`. Predikáty jsou uloženy (a vyhodnocovány) v následujícím pořadí: katalogové číslo, bounding box, typ primitiva, hodnota atributu a vzdálenost. Cílem je seřadit vyhodnocování tak, aby se množina potenciálních výsledků co nejrychleji redukovala.

Příklad spuštění tohoto modulu přes uživatelské rozhraní pro dotaz na objekty typu multilinie (`-t 3`) a vzdálenost menší než $1000\,m$ od objektu s katalogovým číslem 54:
`./select -i layerIn.db -o layerOut.db -t 3 -d "54 < 1000"`

Vyhodnocení bounding boxu V tomto případě se využívá prostorového indexu. Algoritmus je takový, že se vytvoří polygon ve tvaru bounding boxu a ten se vloží do struktury indexu. Následně se vyberou všichni sousedé a porovnají se na příslušnost ohraničení.

Vyhodnocení vzdálenosti Výpočet vzdálenosti dvou geometrických objektů implementuje třída `Distance`. Použitým algoritmem je algoritmus založený na *Gilbert - Johnson - Keerthi* metodě. Princip algoritmu je popsán v literatuře [10]. Nevýhodou této metody je

to, že pracuje pouze s konvexními objekty. Pokud je nějaký objekt nekonvexní, počítá se vzdálenost k jeho konvexní obálce.

4.7.3 Implementace třídy pro testování překryvu polygonů

Třída `GeometryInPolygon` je velmi často používána vektorovými analýzami. Je založena na testování příslušnosti bodu polygonu. Tuto operaci implementuje třída `PointInPolygon` a to metodou *vrhání paprsku*. Třída `GeometryInPolygon` poskytuje tři metody, `inside()`, `outside`, `overleap()`. Tyto metody postupně testují objekty na *obsažení*, *rozdělitelnost* a *překryv*. V nejlepším případě je složitost výpočtu u všech metod $O(n)$, v nejhorším případě $O(n * m)$, kde n je počet hraničních bodů prvního polygonu a m počet hraničních bodů druhého polygonu.

4.7.4 Implementace modulu pro výpočet konvexních obálek

Tento modul implementuje třída `ConvexHull`. Využívá se k tomu rekurzivního algoritmu 8, který je pro svoji podobnost s *quick sortem* nazýván *quick hull*. Tato metoda je založena na přístup *Rozděl a panuj*. Průměrná časová náročnost je $O(n * log(n))$, v nejhorším případě je $O(n^2)$.

Operace výpočtu konvexní obálky se provede pro všechny geografické objekty vrstvy. Výpočtem mohou vzniknout nevalidní data, kdy se výsledné polygony překrývají. Proto je nutná použít operace modulu *sjednocení*, viz. sekce 4.7.5, které z překrývajících se objektů utvoří jeden.

Algoritmus 8: Pseudokód algoritmu pro výpočet konvexní obálky metodou *quick hull*

Input: Množina bodů `N`
Output: Konvexní obálka `CH`

```
1 p0 := nalezení bodu z N s nejmenší x-souřadnicí;
2 p1 := nalezení bodu z N s největší x-souřadnicí;
3 CH := přidej p0 a p1;
4 l0 := nalezení nejvzdálenějšího bodu z N od přímky p0p1 v levé polorovině;
5 r0 := nalezení nejvzdálenějšího bodu z N od přímky p0p1 v pravé polorovině;
6 CH := přidej l0 a r0;
7 rekurzivně hledej nejvzdálenější body od úseček p0l0, l0p1, p1r0, r0p0;
8 return CH;
```

4.7.5 Implementace modulu pro překryvové analýzy

Tento modul implementuje dvě operace překryvových analýz. Jsou jimi *sjednocení* a *průnik*. Funkcionalitu modulu implementuje třída `VOverlay`. V této sekci budou následně popsány implementované metody pro užití operací sjednocení a průniku nad celou vrstvou.

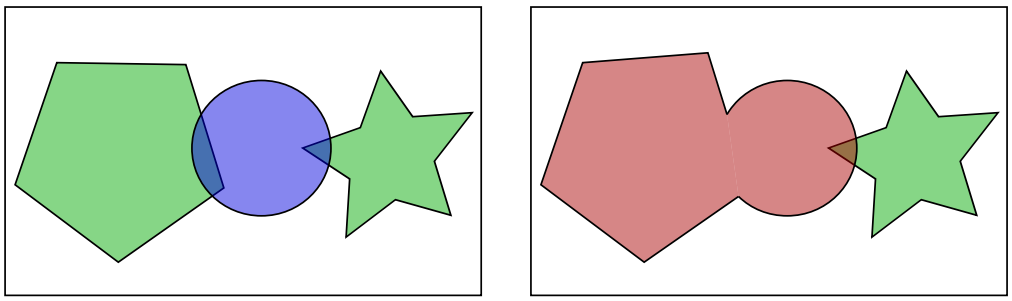
V kapitole 3 jsou pak popsány algoritmy pro sjednocení (průnik) dvou geometrických objektů.

Implementace operace sjednocení vrstev (vrstvy)

Sjednocení celé vektorové vrstvy je zajímavý problém. Mohou totiž nastat situace, kdy nelze sjednotit objekt z vrstvy první s objektem z vrstvy druhé a tento výsledek považovat za nový objekt výstupní vrstvy. Příklad, kdy takováto situace nastává je demonstrována na obrázku 4.5.

Kvůli takovýmto případům byla implementována metoda takzvaného inkrementálního sjednocování. Princip je vysvětlen na algoritmu 9. Vysvětlený algoritmus je podán v naivní formě. V kódu knihovny jsou implementované optimalizace, které snižují časovou složitost algoritmu. Algoritmus v naivní i optimalizované formě spadá do polynomiální třídy složitosti.

Operace sjednocení je implementována tak, že v případě dodání pouze jedné vrstvy na vstup se vypočítá sjednocení pouze nad jednou vrstvou. Tato funkcionalita je důležitá pro ostatní typy vektorových analýz.



Obrázek 4.5: Ukázka problematického případu při sjednocení vrstev

Algoritmus 9: Pseudokód algoritmu pro sjednocení vrstev

Input: Vrstva `l0`
Input: Vrstva `l1`
Output: Vrstva po operaci sjednocení `lOut`

```
cluster01 := sloučený seznam geografických objektů vrstev l0, l1;
foreach gOb0, gOb1 in cluster01 do
    if gOb0 překrývá gOb1 then
        union := sjednoť objekty gOb0, gOb1;
        odeber gOb0, gOb1 z cluster01;
        vlož union na konec cluster01;
    end
end
lOut := na základě cluster01 vytvoř novou vrstvu;
return lOut;
```

Implementace operace průniku vrstev

Tato metoda vypočítá geometrický průnik dvou vektorových vrstev. Předpokladem pro tuto analýzu je, že žádné dva objekty se nepřekrývají v jedné vrstvě. Nutno podotknout,

že pokud by se objekty překrývaly, jednalo by se o nevalidní geografickou vrstvu a pro tu není chování operací specifikované.

Na rozdíl od operace sjednocení, operace průniku má vždy časovou složitost kvadratickou. Tento údaj vychází z implementovaného algoritmu 10.

Algoritmus 10: Pseudokód algoritmu pro průnik vrstev

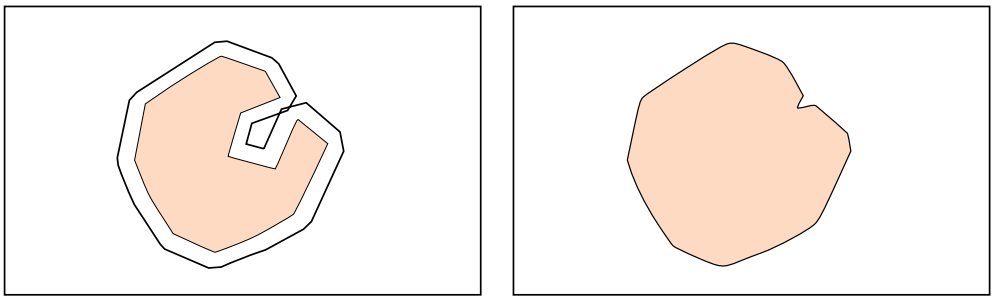
```
Input: Vrstva l0
Input: Vrstva l1
Output: Vrstva po operaci sjednocení lOut
foreach gOb0 : l0 do
    foreach gOb1 : l1 do
        if gOb0 překrývá gOb1 then
            switch typ překryvu do
                case gOb0 obsahuje gOb1
                    přidej gOb1 do lOut;
                end
                case gOb1 obsahuje gOb0
                    přidej gOb0 do lOut;
                end
                otherwise
                    g := vypočítej průnik gOb0, gOb1;
                    přidej g do lOut;
                end
            endsw
        end
    end
end
return lOut;
```

4.7.6 Implementace modulu pro vzdálenostní analýzy

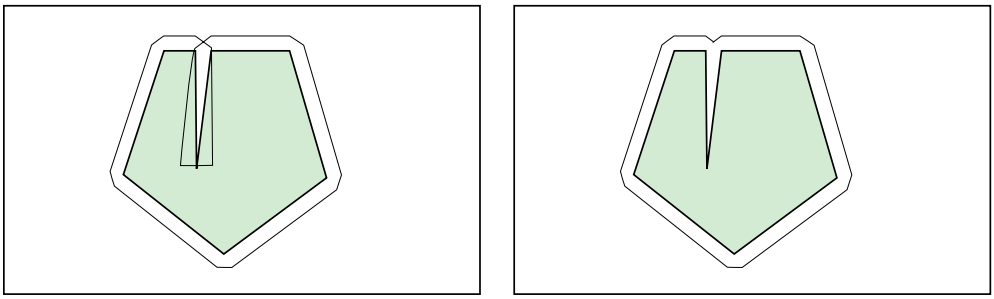
Operace vzdálenostní analýzy implementuje třída `VBufferBuilder`. Pro výpočet operace `buffer` nad vektorovou vrstvou je třeba lineárním průchodem aplikovat operátor vzdálenostní analýzy na každý geografický objekt. Stejně jako v případě modulu pro konvexní obálku, i zde je potřeba po vykonání výpočtu použít metody modulu pro překryvové analýzy. V případě, že se dva a více polygonů vytvořených operací `buffer` překrývají, sloučí se do jednoho polygonu.

V závislosti na konkrétní implementaci knihovny, kdy systém nepodporuje polygony s dírami, může výpočet operace `buffer` nabývat jiného výsledku než systémy díry podporující. Příklad je uveden na obrázku 4.6.

V kapitole 3 byla zmíněna nutnost provádět korekce polygonů po provedení operace `buffer`. Algoritmus 11 popisuje implementované řešení korekce. Znázornění výsledku korekce je na orbázku 4.7.



Obrázek 4.6: Ukázka problematického případu pro operaci `buffer`



Obrázek 4.7: Provedení korekce polygonu

Algoritmus 11: Pseudokód algoritmu korekce polygonu po operaci `buffer`

Input: Polygon `P` po operaci `buffer`
Output: Korektní polygon `PK`

```
1 foreach edge : P do
2 |   if edge protíná edge+1 then
3 |   |   napoj průsečík na hrany;
4 |   end
5 |   else
6 |   |   napoj hrany na sebe;
7 |   end
8 end
9 return lOut;
```

4.7.7 Implementace modulu pro výpočet topologie

Náplň tohoto modulu je provedení výpočtu topologie a sjednocení vektorových objektů, které mají shodné souřadnice. Ukázka případu, kdy není nad vrstvou provedena analýza topologie, je na obrázku 4.1.

Algoritmus pro výpočet topologie je popsán pseudokódem 12. Je založen na sloučení geografických objektů se shodnými souřadnicemi pomocí *sjednocovací tabulky*. Současná verze implementace podporuje výpočet na základě prostorového indexu. Experimentálně bylo zjištěno, že pro vrstvu s 3000 objekty byla doba výpočtu zkrácena na polovinu (z 22 s na 10 s) oproti výpočtu bez indexu.

Součástí topologického modulu je i výpočet délky jednotlivých linií. Postup výpočtu je popsán algoritmem 13.

Algoritmus 12: Pseudokód algoritmu pro výpočet topologie vrstvy (in sito)

```
   Input: Vrstva L
   Output: Vrstva s vypočítanou topologií
 1 table := tabulka sloučení objektů;
 2 begin Sloučení objektů dle souřadnic
 3 |   for index : terminalIndexes do
 4 |   |   for gOb0, gOb1 : index do
 5 |   |   |   if geometrie gOb0 = geometrie gOb1 then  gOb1 := gOb0;
 6 |   |   |   poznač v table; ;
 7 |   |   end
 8 |   end
 9 end
10 begin Vytvoření topologických vazeb
11 |   foreach gOb0, gOb1 : L do
12 |   |   catIds0 := katalogová čísla objektů obsažených v gOb0;
13 |   |   catIds1 := katalogová čísla objektů obsažených v gOb1;
14 |   |   catIds01 := catIds0 průnik catIds1;
15 |   |   if catIds01 je neprázdné then
16 |   |   |   gOb0 := přidej do topologie gOb1;
17 |   |   |   gOb1 := přidej do topologie gOb0;
18 |   |   end
19 |   end
20 end
```

Algoritmus 13: Pseudokód algoritmu pro výpočet délek linií (in sito)

Input: Vrstva s topologií `L`
Output: Vrstva s atributy délky `L`

```
1  vytvoř meta atribut segmentLength;
2  foreach point : L.points do
3  |  point.segmentLength := 0;
4  end
5  foreach line : L.lines do
6  |  line.segmentLength := součet vzdáleností mezi body tvořícími linii;
7  end
8  foreach mline : L.mlines do
9  |  mline.segmentLength := součet délky linií tvořící multilinii;
10 end
11 foreach polygon : L.polygons do
12 |  polygon.segmentLength := součet délky multilinií tvořící polygon;
13 end
```

Kapitola 5

Testování knihovny La'GIS

Tato kapitola popisuje testování jednotlivých modulů knihovny La'GIS. Výsledky jsou zobrazeny formou obrázků vrstvy před a po provedení analýzy.

Některé algoritmy nepracují naprosto korektně nad reálnou geografickou vrstvou, proto jsou přiloženy i ukázky na tzv. demovrstvách.

5.1 Vektorové vrstvy použité pro analýzu

V následujících sekcích jsou použity vektorové vrstvy, které jsou zobrazeny na obrázcích 5.1. V testovaných příkladech jsou tyto mapy pojmenovány *Frida*, *Roads*, *States* a *London*.

5.1.1 Použití indexační struktury typu Quadtree

Mnoho z následujících analýz využívá rozčlenění vrstvy do indexační struktury. Na obrázku 5.2 je znázorněna struktura konkrétního stromu pro vrstvu *Frida*.

5.2 Testování selekční analýzy

Tato část obsahuje výsledky testování selekční analýzy. Jedná se o operace vyhledávání dle všech zadaných predikátů. Predikáty jsou pro názornost zadávány postupně.

Selekční analýza dle atributů

Na obrázku 5.3 je výsledek výpočtu pro selekci podle atributu. Konkrétním dotazem zde bylo vyhledání všech cest, které jsou kratší než sto metrů. Výpočet se prováděl nad vrstvou Frida, pro kterou již byla provedena topologická analýza (čili délka cest je uložena formou atributů). Analýza byla spuštěna s následujícími parametry:

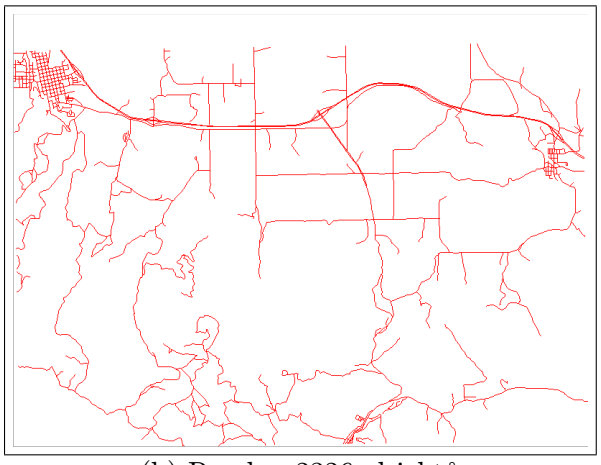
```
./select -i ../tmp/frida_streets_t.db -a "segmentLength < 100" -s
```

Selekční analýza dle vzdálenosti

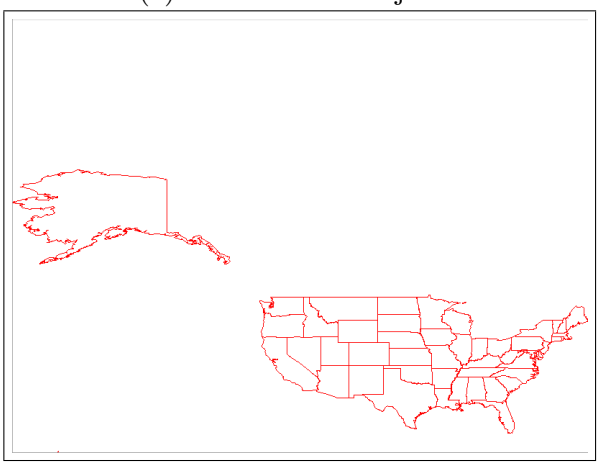
Ukázkou selekční analýzy podle vzdálenosti je vektorová vrstva znázorněná na obrázku 5.4. Výpočet probíhal nad vrstvou States. Výsledek reprezentuje všechny státy, které jsou vzdáleny od zvýrazněného bodu (katalogové číslo 120) do určené vzdálenosti. Příklad spuštění analýzy:

```
./select -i ../tmp/states.db -d "120 < 5" -s
```

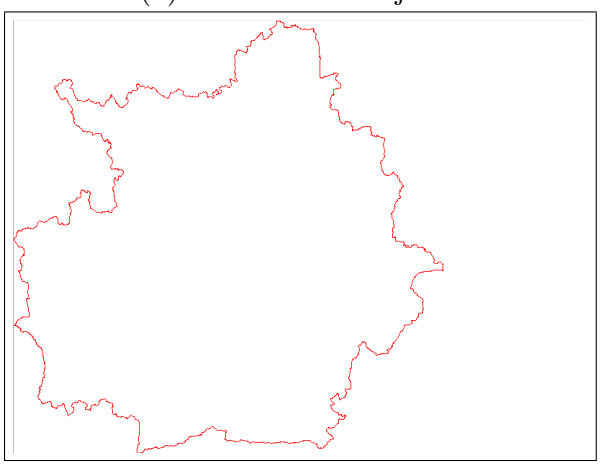
(a) Frida - 33584 objektů



(b) Roads - 2326 objektů

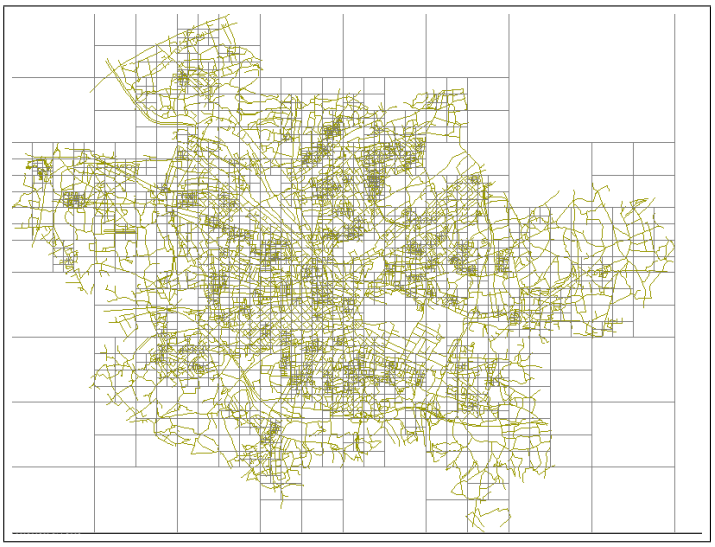


(c) States - 204 objektů

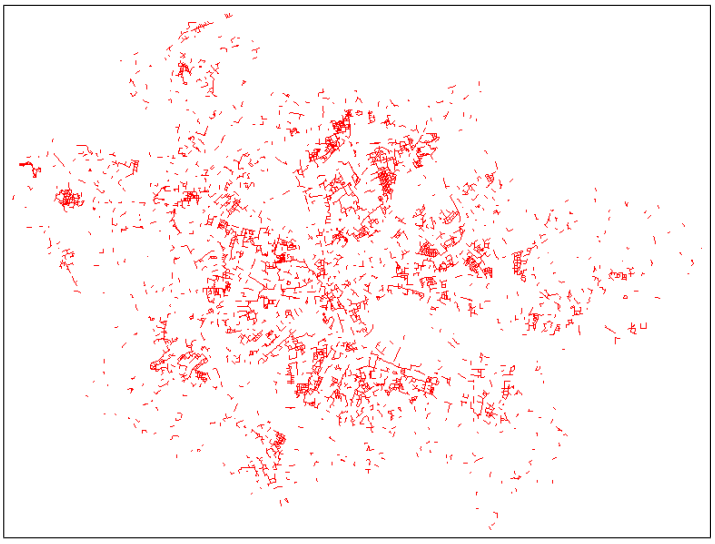


(d) London - 4 objekty

Obrázek 5.1: Originální vektorové vrstvy



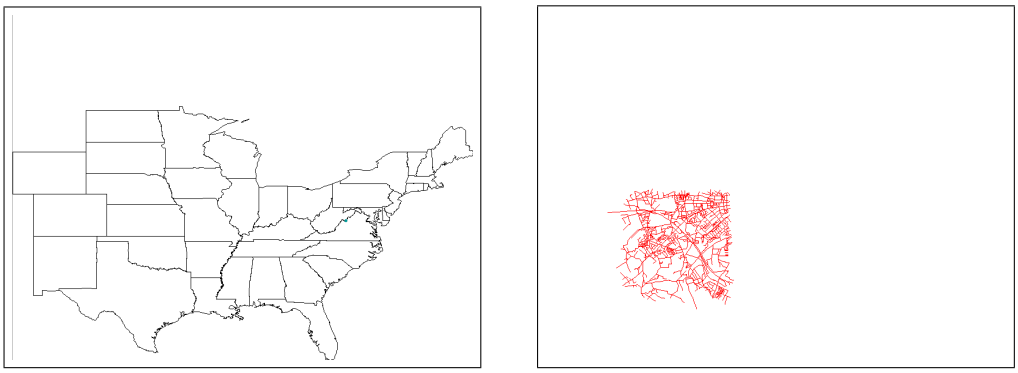
Obrázek 5.2: Rozčlenění vrstvy *Frida* do indexační struktury



Obrázek 5.3: Selekce cest kratších než sto metrů

Selekční analýza dle bounding boxu

Výstupem této operace je seznam objektů, které spadají do vybrané oblasti. Výběrová oblast se zadává formou levého dolního a pravého horního rohu. Pro užití analýzy je třeba si vypsat informace o vrstvě, odkud zjistíme levý dolní roh vrstvy a její šířku a délku. Příklad analýzy je na obrázku 5.4 a bylo ho dosaženo spuštěním modulu s parametry:
`./select -i ../tmp/frida_streets_t.db -b "3430000 5790000 3434000 5794000" -s`



(a) Selekce objektů podle vzdálenosti (b) Selekce objektů podle bounding boxu

Obrázek 5.4: Příklady selekční analýzy nad vrstvami (a) States a (b) Frida

5.3 Testování překryvových analýz

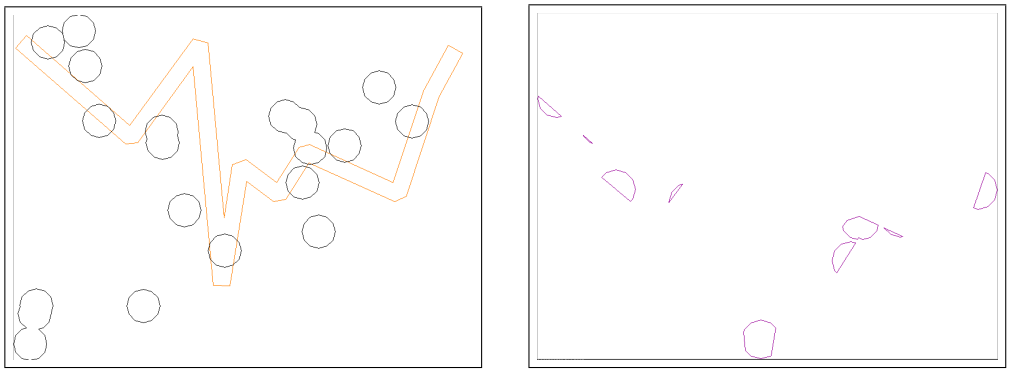
Překryvové analýzy jsou pro testování na reálných datech velmi náročné na výpočet. Mnohdy algoritmus nepodává korektní výsledky. Je to dáno komplikovanými tvary polygonů, velkým počtem objektů a podobně. Pro následující operace jsou tak použity demovrstvy, na kterých je testován implementovaný algoritmus.

Překryvová analýza s operátorem průniku

Tato analýza vypočítá geometrický průnik objektů dvou vrstev. Na testovaných vrstvách *frida_green.db* a *frida_water.db* jsou však průniky pro zobrazení velmi malé. Výsledek lze však získat spuštěním aplikace:

```
./overlay -1 ../tmp/frida_green.db -2 ../tmp/frida_water.db -r AND -s
```

Funkčnost algoritmu je tak ukázána na demovrstvě. Vstupy a výstupy výpočtu jsou zobrazeny na obrázku 5.5.

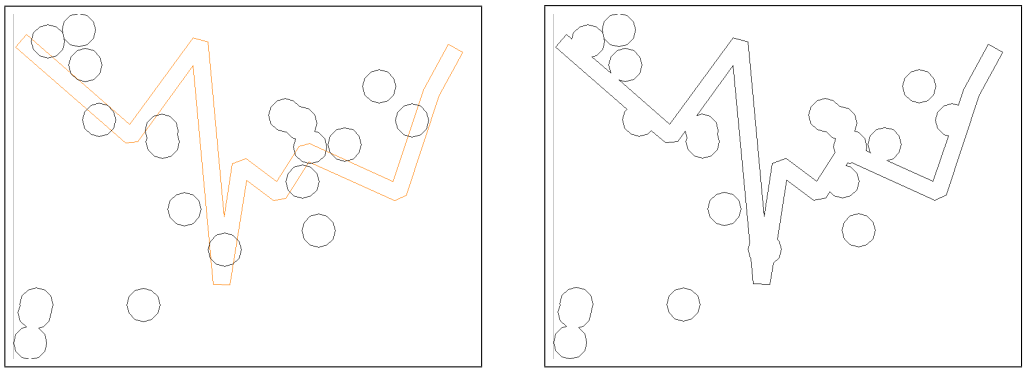


(a) Vstupní vrstvy pro analýzu průniku (b) Výstup analýzy průniku

Obrázek 5.5: Testovaný algoritmus průniku vrstev na demonstračních datech

Překryvová analýza s operátorem sjednocení

Tato operace je mírně pomalejší než operace průniku. To je dáno rozdílným algoritmem pro výpočet nové vrstvy. K provedení analýzy jsou zapotřebí dvě vektorové vrstvy. Výsledek operace je zobrazen na obrázku 5.6.



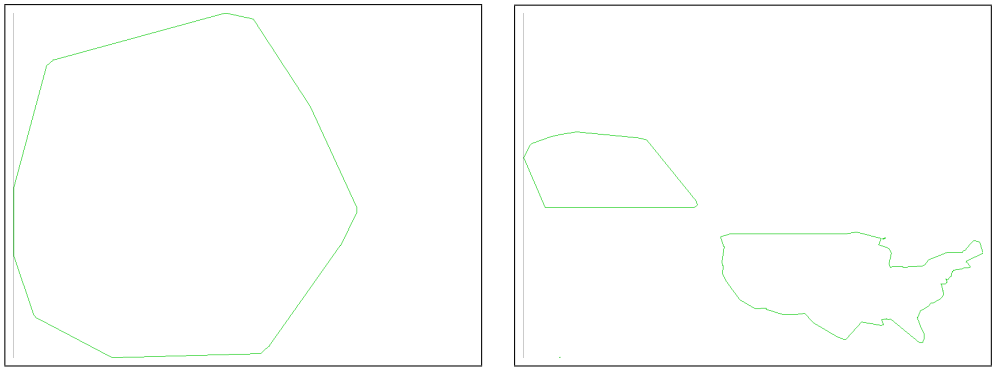
(a) Vstupní vrstvy pro analýzu sjednocení (b) Výstup analýzy sjednocení

Obrázek 5.6: Testovaný algoritmus sjednocení vrstev na demonstračních datech

5.4 Testování modulu pro výpočet konvexní obálky

Konvexní obálka vrstvy znamená výpočet konvexní obálky každého objektu a jejich sjednocení. Za tímto účelem je použit modul pro sjednocení (výpočtem konvexních obálek objektů

se polygony zjednodušily, čili je možné použít sjednocení). Výsledek aplikovaného vypočtu je zobrazen na obrázku 5.7.

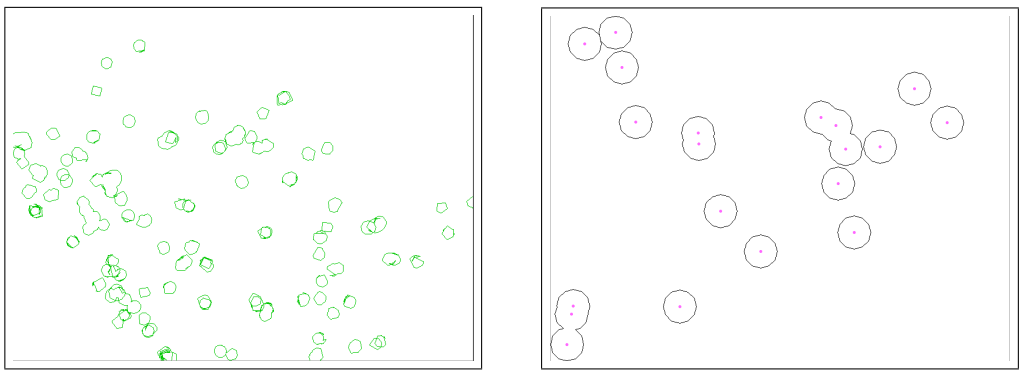


(a) *London* (b) *States*

Obrázek 5.7: Výpočet konvexní obálky pro referenční vrstvy *London* a *States*

5.5 Testování vzdálenostní analýzy

Výpočet vzdálenostní operace `buffer` má pro složité geografické podklady nekorektní výsledky. Je to dáno složitostí metody pro korekci vypočítaného polygonu. Téměř vždy se vyskytne případ polygonu, pro který není implementovaná metoda dostatečně robustní. Na příkladu 5.8 je znázorněno aplikování operace `buffer` na různé vektorové vrstvy. Z obrázku je patrné využití modulu překryvové analýzy - sjednocení. Příklad spuštění analýzy:
`./buffer -i ../points20.db -d 30 -s`



(a) Provedení operace nad vrstvou *Water* (b) Provedení operace nad body

Obrázek 5.8: Výpočet vzdálenostní analýzy `buffer`

5.6 Testování modulu pro výpočet topologie

Výpočet topologie se nijak neprojeví na vzhledu geografických dat. V průběhu vypočtu se však může značně měnit vnitřní struktura a vztahy objektů. Následující příklad spuštění topologické analýzy ukazuje, o kolik je v nové vrstvě méně objektů, které tvoří geografickou vrstvu. Tento poměr závisí na počtu bodů, které mají linie společné.
`> ./topology -i ../tmp/roads.db`

```
> Opening..
> Creating topology..
> Compression: 30 %
> Adding length informations..
```

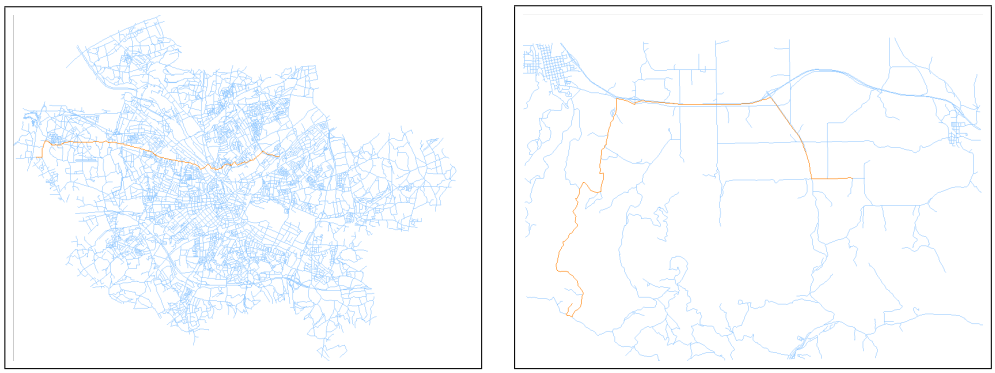
Výpočet topologie velmi závisí na velikosti vrstvy, resp. na počtu jejích objektů. Pro vrstvu *Roads* trvá výpočet kolem 2 *s*, pro vrstvu *Frida* se dostáváme do řádu minut. Analýzu lze provádět pouze nad vrstvou, nad kterou nebyla topologie vypočítaná. Vždy se vytváří nový atribut délek linií. Ten má využití v následujících analýzách.

5.7 Testování síťových analýz

Provádění síťových analýz má smysl pouze nad takovými vektorovými vrstvami, které reprezentují například síť silnic, vodních toků, kanalizace, elektrického vedení a podobně. Následující operace jsou prováděny nad vrstvami *Frida* a *Roads*, které reprezentují síť pozemních komunikací.

5.7.1 Síťová analýza výpočtu nejkratší cesty

Tato operace je používána v ostatních síťových analýzách, proto je důležité, aby byla rychlost výpočtu cesty co největší. V následujících ukázkách 5.9 je zobrazen rozdíl rychlosti výpočtu mezi vrstvami *Frida* a *Roads*. Vrstvy se velmi liší v počtu uzlů, které obsahují. Zároveň se však liší i délka cesty, proto naměřené výsledky lze brát pouze orientačně.



(a) Výpočet trval 0.841 *s* (b) Výpočet trval 0.115 *s*

Obrázek 5.9: Výpočet nejkratší cesty z `1268` do `2228` ve vrstvách *Frida* a *Roads*

5.7.2 Síťová analýza rozdělení zdrojů

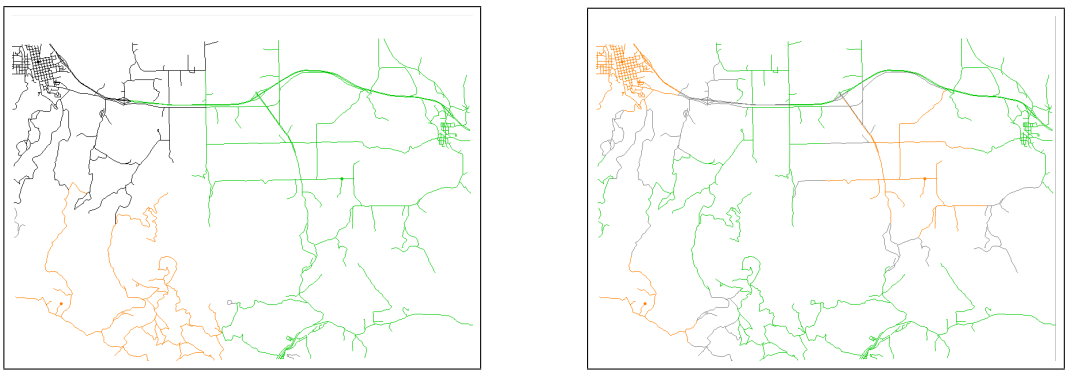
Výpočet operace rozdělení zdrojů je založen na testování všech uzlů sítě a počítání jejich nejkratší cesty k centrům. Následující testy byly provedeny pouze nad vrstvou *Roads*. Pro vrstvu *Frida* trval výpočet nepřiměřeně dlouho. Výsledek analýzy je na obrázku 5.10. Příklad spuštění analýzy rozdělení zdrojů:

```
./alloc -i ../test/roads_t.db -1 1268 -2 2228 -3 75 -s
```

5.7.3 Síťová analýza obslužných zón

Stejně jako předchozí operace 5.7.2 je i tato analýza založena na výpočtech nejkratší cesty. Ze stejného důvodu jsou proto testy provedeny pouze na vrstvě *Roads*. Příklad analýzy je na obrázku 5.10. Spuštění analýzy obslužných zón:

```
./iso -i ../test/roads_t.db -1 1268 -2 2228 -3 75 -s
```



(a) Výpočet rozdělení zdrojů (b) Výpočet obslužných zón s poloměrem 3000 m

Obrázek 5.10: Síťové analýzy nad vrstvou *Roads*

Kapitola 6

Závěr

Cílem této diplomové práce bylo navrhnout knihovnu La'GIS, která implementuje operace vektorové analýzy. Důraz byl kladen na rozčlenění knihovny do samostatných modulů, které však mohou využívat navzájem své funkce. Součástí implementace je vytvoření programátorského a uživatelského rozhraní pro používání jednotlivých modulů knihovny.

Navržený subsystém je dostatečně robustní na to, aby se v případných dalších verzích dal snadno rozšířit o nová geometrická primitiva a operace nad nimi. Systém ukládání dat na disk je obalen rozhraním abstraktní třídy, čili je možné implementovat podporu jiného formátu než použitého `sqlite3`. Knihovna La'GIS je napojitelná do dalšího geografického systému přes rozhraní formátu *shapefile*. Programátorské rozhraní na analytické moduly má jednotné chování, čili implementace nových algoritmů tak může snadno vycházet z již hotových součástí.

Implementace analytických algoritmů značně zpomalila jejich efektivitu. Byť některé algoritmy mají lineární či kvadratickou složitost, ve výsledku je tato složitost větší, protože se krom geografických objektů prohledávají i body představující jejich geometrii. Většina operací tak nepracuje s abstraktním geografickým objektem, ale s množinou jeho souřadnic.

Výpočetní výkon knihovny nebyl nijak velký, proto byla implementována sada optimalizačních prostředků. Nejspíše nejvýkonnější z nich je používání prostorového indexu. Velká část analýz je na něm založena a tak díky preprocesingu je ušetřeno spoustu výpočetního času. Současně se analýzy provádějí pouze nad konečnými geografickými objekty, nikoliv nad jejich implementačními částmi.

Jak je psáno výše, struktura programu je implementována pro snadné navázání práce a rozšíření funkčnosti knihovny. Za úvahu by stála změna typu úložiště geografických objektů v dynamické paměti. Současný přístup je implementován formou asociativního pole. Jelikož ve většině případů je posloupnost katalogových čísel spojitá, bylo by vhodnější implementovat sklad formou pole s přímým přístupem. Taktéž by bylo vhodné vybudovat nad jednotlivými geografickými objekty strukturu metadat, která obsahují například obálku objektu. V současnosti jsou tyto informace počítány, což snižuje výpočetní výkon knihovny.

Implementace geografického informačního systému je záležitost nečekaně velká a složitá. Všechny komponenty se na sebe nabalují a nenechávají prostor pro chyby. Jsem rád, že jsem měl příležitost nahlédnout do vnitřní stavby GIS a prozkoumat metody a algoritmy, které se používají pro vektorovou analýzu.

Literatura

[1] Geospatial Data Abstraction Library.
URL http://www.gdal.org

[2] Informace o civilním geodetickém systému S-JTSK.
URL http://krovak.webpark.cz/

[3] Oficiální stránky geografického sysému ArcGIS.
URL http://resources.arcgis.com/en/help/getting-started/articles/026n00000014000000.htm

[4] Oficiální stránky geografického sysému OpenJump.
URL http://www.openjump.org/

[5] Oficiální stránky geografického sysému QGIS.
URL http://docs.qgis.org/

[6] Oficiální stránky geografického systému GRASS.
URL http://grass.osgeo.org/

[7] Standard ESRI - shapefile.
URL http://dl.maptools.org/dl/shapelib/shapefile.pdf

[8] Bernhardsen, T.: *Geographic Information Systems: An Introduction*. Wiley, 2002, ISBN 9780471419686.
URL http://books.google.cz/books?id=e-yvDHkDLJQC

[9] Clarke, K. C.: *Getting Started with Geographic Information Systems*. Pearson Education, Inc., 2003, ISBN 0-13-046027-3.

[10] Ericson, C.: *The Gilbert - Johnson - Keerthi algorithm*.
URL https://www.fit.vutbr.cz/study/courses/VGE/private/reading/2/SIGGRAPH04_Ericson_GJK_notes.pdf

[11] Hrubý, M.: *Geografické Informační Systémy (GIS) Studijní Opora*. 2006.
URL
http://perchta.fit.vutbr.cz:8000/vyuka-gis/uploads/1/GIS-final2.1.pdf

[12] Korte, G. B.: *The GIS Book*. OnWord Press, páté vydání, 2001, ISBN 0-7668-2820-4.

[13] Maguire, D. J.; Goodchild, M.; Rhind, D.: An overview and definition of GIS. *Geographical information systems: Principles and applications*, ročník 1, 1991: s. 9–20.

[14] Malczewski, J.: *GIS and Multicriteria Decision Analysis*. Wiley, 1999, ISBN 9780471329442.
URL http://www.google.cz/books?id=2Zd54x4_2Z8C

[15] Peuquet, D.; Marble, D.: *Introductory Readings In Geographic Information Systems*. Taylor & Francis, 2003, ISBN 9780203393246.
URL http://www.google.cz/books?id=_y5Dk7NjEBoC

[16] Rainsford, H. F.: Long geodesic on the ellipsoid.
URL https://faculty.unlv.edu/jensen/CEE_121/pdf/LongGeodesicsOnTheEllipsoid-HFRainsford.pdf

[17] Sickle, J. V.: *Basic GIS Coordinates*. CRC Press, 2004, ISBN 0-915-30216-1.