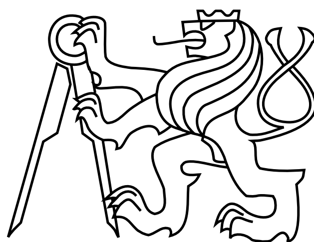
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ
OBOR GEOMATIKA



DIPLOMOVÁ PRÁCE

POLYGONIZACE LOMENÝCH ČAR ZE ŠPAGETOVÉHO MODELU

Vedoucí práce: Ing. Martin Landa, PhD.
Katedra geomatiky

červen 2020

Bc. Tomáš KLEMSA

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Thákurova 7, 166 29 Praha 6

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Bc. Klemsa Jméno: Tomáš Osobní číslo: 439229

Zadávající katedra: Katedra geomatiky

Studijní program: Geodézie a kartografie

Studijní obor: Geomatika

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Polygonizace lomených čar ze špagetového modelu

Název diplomové práce anglicky: Polygonization of LineString Spaghetti Data

Pokyny pro vypracování:
Cílem diplomové práce je navrhnout algoritmus pro vytvoření polygonových prvků na základě vstupních linií uložených ve špagetovém modelu. Tento algoritmus vhodně implementovat pro potřeby služby pátrání a záchrany (Search and Rescue) hledaných osob. Algoritmus bude brát v potaz objem a charakter vstupních dat jako je např. výskyt fuzzy průsečíků a přesahů linií (dangels).

Seznam doporučené literatury:
de Berg, van Kreveld, Overmars M., Schwarzkopf O.: Computational Geometry, 2000, Springer
Bayer T.: Algoritmy v digitální kartografii, 2008, UK v Praze

Jméno vedoucího diplomové práce: Ing. Martin Landa, PhD.

Datum zadání diplomové práce: 20.2.2020 Termín odevzdání diplomové práce: 17.5.2020
Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku

Podpis vedoucího práce Podpis vedoucího katedry

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.

Datum převzetí zadání Podpis studenta(ky)

ABSTRAKT

Patřičné mapové podklady jsou základem pro pátrání po pohřešovaných osobách. Velký prostor je rozdělen liniemi na sektory, ve kterých samotné pátrání probíhá. Tato práce se zabývá tvorbou polygonů(sektorů) z množiny linií (tzv. polygonizací). V první části práce je provedena rešerše algoritmů a aktuálně dostupných konvenčních nástrojů. Následně je implementován polygonizační nástroj za pomoci knihovny Java Topology Suite.

KLÍČOVÁ SLOVA

Polygonizace, špagetový model, výpočetní geometrie, Java, Java Topology Suite

ABSTRACT

The basis for the missing persons search are appropriate map data. The large space is divided by linestring into sectors in which the search itself takes place. This work deals with the creation of polygons (sectors) from a set of linestring (so-called polygonization). In the first part of the work, a search of algorithms and currently available conventional tools is done. Subsequently, a polygonization tool is implemented using the Java Topology Suite library.

KEYWORDS

Polygonization, spaghetti model, computational geometry, Java, Java Topology Suite

PROHLÁŠENÍ

Prohlašuji, že diplomovou práci na téma „Polygonizace lomených čar ze špagetového modelu“ jsem vypracoval samostatně. Použitou literaturu a podkladové materiály uvádím v seznamu zdrojů.

V Praze dne ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce Ing. Martinu Landovi, Ph.D. za rady a podporu. Dále bych rád poděkoval Ing. Janu Růžičkovi, Ph.D. za poskytnutí dat a rady.

Obsah

1 Úvod **8**

1.1 Obsah práce . . . 8

2 Projekt Pátrač **10**

2.1 Pátrání po pohřešovaných osobách . . . 10

2.1.1 Základní pojmy . . . 11

3 Vektorové datové modely **12**

3.1 Špagetový model . . . 12

3.2 Topologické modely . . . 13

4 Navrhování algoritmů **15**

4.1 Časová a prostorová složitost algoritmů . . . 15

4.1.1 Asymptotická složitost . . . 15

4.1.2 Stanovení časové složitosti . . . 17

4.2 Metody algoritmizace ve výpočetní geometrii . . . 17

4.2.1 Metoda hrubé síly . . . 18

4.2.2 Heuristické algoritmy . . . 18

4.2.3 Rozděl a panuj . . . 18

4.2.4 Zametací přímka . . . 19

4.2.5 Inkrementální algoritmy . . . 20

5 Rešerše používaných algoritmů **22**

5.1 Výpočet průsečíků množiny linií . . . 22

5.1.1 Vzájemná poloha dvou úseček . . . 22

5.1.2 Nalezení průsečíku dvou úseček . . . 23

5.1.3 Bentley–Ottmannův algoritmus . . . 26

5.2 Tvorba polygonů z množiny linií . . . 26

6 Konvenční prostředky a polygonizace **28**

6.1 ArcGIS Desktop . . . 28

6.2 QGIS Desktop . . . 28

6.3 PostGIS . . . 29

7 Návrh a implementace **32**
7.1 GeoTools 32
7.2 JTS Topology Suite 32
7.3 Návrh 33
7.4 Implementace 34
7.4.1 Třída PolygonizeOp 35
7.5 Příklad použití 35

8 Závěr **36**

Seznam zkratek **37**

Literatura **39**

A Srovnání polygonizace v ArcGIS a QGIS **42**
A.1 Parametry počítače 42
A.2 Výsledky a porovnání 42

B Třída PolygonizeOp **44**

C Elektornické přílohy **49**

1 Úvod

V životě každého z nás se pravděpodobně někdy přihodilo, že jme se zmátli své orientační smysly a ocitli se na jiném místě, než jsme čekali. Často můžeme zabloudit, pokud nevidíme v dáli žádný pevný bod, například se pohybujeme v mlze. V takovém případě nás lidská mysl dokáže dokonale zmást a i když si myslíme, že se pohybujeme po přímce, pokud bychom se koukli na naše stopy shora, budou připomínat spíše zamotaný provaz. Pokud se v takovémto případě dostaneme do potíží, může naše zdraví nebo život záviset na času, ve kterém nás záchranné složky dokáží vypátrat, zejména jedná-li se o osoby závislé na cizí péči. Pro takovéto případy je důležité mít plán postupu při pátrání po pohřešovaných osobách. Jedná-li se o plošné pátrání, je jednou z nejpodstatnějších částí speciálně vyhotovený mapový podklad takzvaných pátracích sektorů. Tyto sektory jsou plochy o vhodném tvaru a velikosti, sloužící pátrací četě pro rozdělení velkého prostoru na menší a tak optimalizují dobu pátrání na minimum.

Tato práce si klade za cíl vytvořit rešerši používaných algoritmů a nástrojů pro tvorbu polygonových prvků z množiny linií. Na základě těchto znalostí se pokusíme navrhnout a implementovat vhodný postup, který bude jednoduše použitelný a zautomatizovatelný v praktickém využití. Tyto vytvořené polygony budou poté dále zpracovávány do polygonů o vhodném tvaru a velikosti, které budou sloužit pro policejní pátrání v terénu. Tématem optimalizace polygonů se zabývá diplomová práce *Aplikace pro tvorbu pátracích sektorů na základě přirozených bariér* [24].

1.1 Obsah práce

Kapitola 2

V této kapitole se zabýváme stručným popisem projektu Pátrač. Budou zde uvedeny základní pojmy související s tímto projektem a naznačen postup pátracích akcí.

Kapitola 3

Jelikož v zadání diplomové práce se hovoří o špagetovém modelu, budou v této kapitole zmíněny základní datové modely v GIS a rozdíly mezi nimi.

Kapitola 4

Kapitola zabývající se algoritmizací ve výpočetní geometrii. Budou zde uvedeny základní techniky návrhu algoritmů, dále zde bude zmíněno hodnocení výkonu algoritmů co se týče časových a paměťových nároků.

Kapitola 5

V této kapitole se začínáme zabývat vlastní polygonizací množin linií. Postupujeme zde od základních geometrických výpočtů až po komplexnější algoritmy.

Kapitola 6

V této části práce zmíníme vybrané konvenční prostředky, které lze pro polygonizaci linií využít, a porovnáme je mezi sebou.

Kapitola 7

Kapitola zabývající se implementací polygonizace špagetového modelu s využitím knihovny JTS Topology Suite.

2 Projekt Pátrač

Projekt Pátrač vznikl ve spolupráci s Policií ČR. Dle elektronické publikace [12] se jedná o *Využití vyspělých technologií a čichových schopností psů pro zvýšení efektivity vyhledávání pohřešovaných osob v terénu.* Projekt lze rozdělit na dvě základní části. **Technická část**, která má za cíl vytvořit software pro přípravu a řízení pátracích akcí, a **biologickou část**, která se zabývá sběrem biologických dat v terénu během pátrání u psů/psovodů [31]. V této kapitole bylo čerpáno z publikací [9, 21, 24, 31].

Technická část je dále rozdělena do čtyř samostatných modulů.

- Zásuvný modul pro přípravu a řízení operace pro QGIS,
- mobilní aplikace pro komunikaci se štábem,
- aplikace pro segmentaci prostoru na základě přirozených bariér,
- aplikace pro generování datového skladu a projektu pro QGIS.

My se v této práci budeme zabývat jednou z částí přípravy segmentace prostoru na základě přirozených bariér. Konkrétně se bude jednat o segmentaci na polygony ze vstupních linií. Tyto polygony jsou pak dále zpracovávány do vhodných velikostí a tvarů. Následným zpracováním polygonů se zabývá diplomová práce.

2.1 Pátrání po pohřešovaných osobách

Podle charakteru prostředí lze záchranné akce rozdělit na několik základních skupin.

- Sutinové vyhledávání,
- plošné vyhledávání,
- vyhledávání na vodních plochách,
- vyhledávání v horském terénu a v lavinách.

Aplikace by měla být využívána především v oblasti plošného vyhledávání. Jedná se o pátrání po pohřešované osobě či osobách na rozsáhlém území, které spadá do kompetence Policie České republiky. Pátrání je prováděno v sektorech, které jsou děleny přirozenými či uměle vytvořenými bariérami. Vyhledávání v takto vymezených oblastech může být prováděno za pomoci kynologických skupin či pátracích rojnic.

2.1.1 Základní pojmy

Pro lepší pochopení dané problematiky zde budou vysvětleny základní pojmy použité v předchozím odstavci.

Pátrání

Pátrání je činnost, která si klade za cíl nalezení hledaného objektu. Objektem v tomto případě může být osoba pohřešovaná, s neznámou totožností či usmrcená. Pátrání však lze vést i za účelem nalezení movitých věcí, kterými mohou být odcizené dopravní prostředky, zbraně a podobně.

Pohřešovaná osoba

Pohřešovaná osoba je osoba, po které bylo vyhlášeno nebo započato pátrání. Tato osoba může být v bezprostředním ohrožení života, proto je nutné osobu v co nejkratším čase vypátrat a provést záchranu. Často se může jednat o osobu, která je závislá na cizí péči, dítě, seniora nebo nemocnou osobu závislou na zdravotní péči. Život ohrožující mohou být i nepříznivé povětrnostní podmínky nebo okolní prostředí.

Pátrací sektor

Je-li známa přibližná lokalita možného výskytu pohřešované osoby, zpravidla je větší prostor rozdělen na menší, takzvané pátrací sektory. Sektory jsou ohraničeny bariérami či jinými hranicemi a dle členitosti terénu a aktuálních podmínek může být volena velikost sektorů. Zároveň by měl mít sektor vhodný tvar pro efektivnější pohyb pátracího týmu.

Bariéra

Bariérou nazýváme člověkem či přírodně vytvořenou překážku, kterou lze jen obtížně překonat a tudíž lze předpokládat, že hledaná osoba tuto bariéru nepřekonala. Kromě bariér dělí pátrací sektory další linie, které mohou napomoct při postupu pátrání a orientaci pátracích týmů v terénu. Přesná skladba těchto linií nám však není známa, jelikož se jedná o citlivá data.

3 Vektorové datové modely

Následující odstavce byly čerpány z publikací [17, 26, 28]. K uchování prostorových dat v GIS jsou používané dva základní typy, a to vektorová a rastrová data. Rastrová data se hojně využívají pro popis spojitého jevu, například nadmořské výšky, nebo pro obrazová data, jako jsou letecké nebo družicové snímky. Každý pixel obsahuje jednu číselnou hodnotu. Jeden rastrový soubor může obsahovat i více vrstev, pak může být fotoplán vyjádřen hodnotami barev RGB, zatímco nadmořská výška bude pravděpodobně vyjádřena jednou číselnou hodnotou.

Druhým typem je vektorové vyjádření. Klasický vektor, tak jak ho známe z matematiky, je charakterizován velikostí a směrem. Lze ho ovšem vyjádřit i dvěma body, čehož využíváme ve vektorové grafice. Odtud je tedy zřejmé, proč vektorová grafika nese toto označení. V GIS se využívají tři základní geometrické útvary. Těmi jsou body, linie a polygony. Tyto základní typy pak slouží pro popis objektů v krajině.

Bod je nejjednodušší geometrický prvek. Nese v sobě informaci o poloze vyjádřenou souřadnicemi X, Y, případně Z.

Linie je vyjádřena posloupností souřadnic. Linie začíná a končí v uzlech, její mezilehlé body se nazývají vertexy. Podle topologických pravidel by linie v jedné vrstvě měly být napojeny pouze přes uzly. Tento topologický koncept se nazývá *konektivita.*

Polygon je vyjádření plochy. Je vyjádřen obdobně jako linie posloupností souřadnic, ovšem počáteční a koncový uzel jsou totožné. To vychází z druhého konceptu topologie, který se nazývá *definice plochy.*

Každý vektorový objekt může kromě geometrické informace obsahovat také různé atributy. Tyto atributy jsou propojeny s objektem pomocí jeho identifikátoru. Liniový prvek vyjadřující pozemní komunikaci proto může obsahovat atributy jako je číslo silnice, maximální povolená rychlost a mnoho dalších, což je značná výhoda oproti rastrovému vyjádření, kde by bylo toto velmi obtížné.

3.1 Špagetový model

Tento model je velice jednoduchý, každý objekt je uložen jako jeden záznam v datové tabulce. Tato tabulka obsahuje identifikátor prvku a jeho souřadnice. Tento model

neukládá žádné prostorové vztahy. Je tedy velice jednoduché přidání či odebrání objektu. Pouze smažeme či přidáme řádek do příslušné tabulky. Tento model je vhodný pro vizualizaci či přenos dat. Nevýhodou je, že společné hranice polygonů jsou v tomto modelu uloženy 2x, pro každý polygon zvlášť. Problém nastává také při některých geometrických analýzách. Jelikož nám nejsou známé žádné prostorové vztahy, musíme například pro zjištění sousedních polygonů projít všechny záznamy v tabulce. Tyto chybějící prostorové vztahy musí být často dopočteny a po provedení analýzy je opět ztrácíme. Pokud tedy nad většími daty chceme provádět často rozsáhlejší prostorové analýzy, tento model není příliš vhodný pro jejich uchovávání.

3.2 Topologické modely

Jak z názvu vyplývá, topologické modely uchovávají informace o topologii. Tedy o prostorových vztazích objektů mezi sebou. Data uložená v topologickém modelu jsou velkou výhodou pro geometrické analýzy, které díky struktuře dat mají mnohé vztahy předpočítané a uložené v datové struktuře. Existuje mnoho způsobů pro uchování topologického modelu. Jako příklad topologického modelu může být formát *GDF/DIME* vyvinutý a používaný v sedmdesátých a osmdesátých letech v USA [29].

Topologie bodů

Topologie bodů je velice jednoduchá a v podstatě koresponduje s uchováním dat ve špagetovém modelu. Jelikož jsou body jeden na druhém nezávislé, nejsou zde uloženy žádné zvláštní topologické vztahy. Jediné, co je o bodech uchováváno, je jejich identifikátor, přes který lze bod propojit se souřadnicemi.

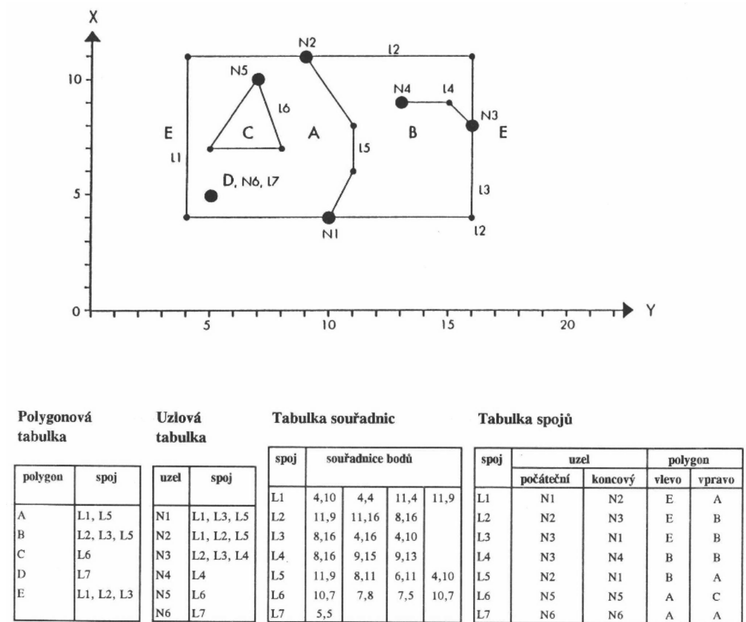
Topologie linií

Pro linie platí pravidlo, že každé dvě linie mohou sdílet pouze uzly, tedy koncové body. Každá část linie je uložena s odkazem na uzly linie. Struktura dále ukládá informace o pravém a levém polygonu vzhledem ke směru linie, pokud takovýto polygon existuje.

Topologie polygonů

Pro každý polygon je definován odkazem na linie, které ho ohraničují. Výhoda v tomto modelu je, že společné linie polygonů jsou uloženy pouze jednou.

Ze struktury dat je vidět, že velice snadno zjistíme například sousední polygony k námi vybranému polygonu. Učinit tak můžeme bez jakéhokoliv přístupu k souřadnicím a geometrickým výpočtům, narozdíl od špagetového modelu. To může ušetřit mnoho času při prostorových analýzách.



Polygonová tabulka

polygon	spoj
A	L1, L5
B	L2, L3, L5
C	L6
D	L7
E	L1, L2, L3

Uzlová tabulka

uzel	spoj
N1	L1, L3, L5
N2	L1, L2, L5
N3	L2, L3, L4
N4	L4
N5	L6
N6	L7

Tabulka souřadnic

spoj	souřadnice bodů			
L1	4,10	4,4	11,4	11,9
L2	11,9	11,16	8,16	
L3	8,16	4,16	4,10	
L4	8,16	9,15	9,13	
L5	11,9	8,11	6,11	4,10
L6	10,7	7,8	7,5	10,7
L7	5,5			

Tabulka spojů

spoj	uzel		polygon	
	počáteční	koncový	vlevo	vpravo
L1	N1	N2	E	A
L2	N2	N3	E	B
L3	N3	N1	E	B
L4	N3	N4	B	B
L5	N2	N1	B	A
L6	N5	N5	A	C
L7	N6	N6	A	A

Obrázek 3.1: Ukázka topologického modelu (převzato z [17])

4 Navrhování algoritmů

V této kapitole se budeme zabývat problematikou návrhu algoritmů ve výpočetní geometrii. Abychom byli schopni jednotlivé algoritmy mezi sebou porovnávat co se týče rychlosti a nároků na paměť, bude zde zmíněna asymptotická složitost. Dále zde budou představeny nejznámější techniky návrhů algoritmů ve výpočetní geometrii, které nám mohou usnadnit práci a získat na problém jiný pohled.

4.1 Časová a prostorová složitost algoritmů

Abychom mohli porovnávat algoritmy řešící stejný problém mezi sebou a rozhodnout se, kdy který algoritmus použít, slouží nám k tomuto účelu základní dvě míry pro porovnání.

- časová složitost
- prostorová složitost

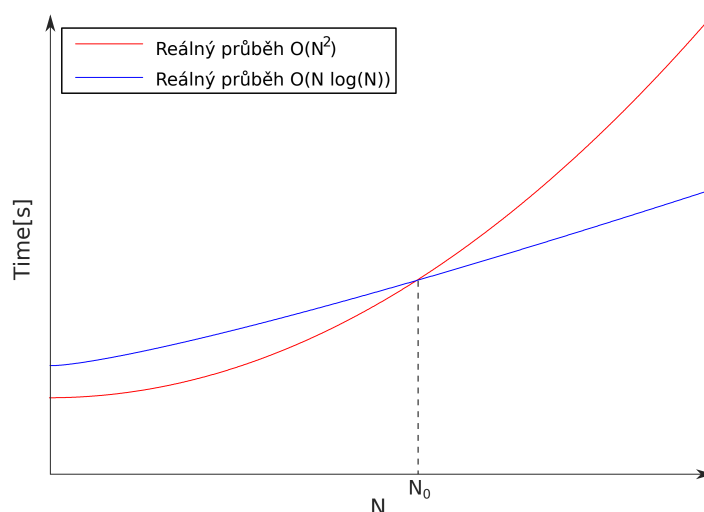
Časová složitost vyjadřuje, jak dlouho bude výpočet podle daného algoritmu trvat, obdobně prostorová složitost nám vyjadřuje, kolik prostoru, tedy paměti, danému algoritmu budeme muset poskytnout pro výpočet. Je očividné, že vyjadřovat časovou složitost v sekundách a prostorovou v bytech by vzhledem k různému hardwaru či různé implementaci nebylo příliš vhodné. Pro popis složitosti algoritmu tedy používáme *asymptotickou složitost*.

4.1.1 Asymptotická složitost

Asymptotická složitost se vyjadřuje jako matematická funkce, popisující závislost využití paměťového prostoru nebo výpočetního výkonu na velikosti vstupních dat N. Důležité je, že tato funkce je neklesající a vyjadřujeme ji pouze jako třídu složitosti, tedy typ funkce, a ne jako přesné vyjádření. Pro příklad funkce $f(x) = 1000N^2$ nebo $g(x) = N^2 + 1000N$ jsou třídy složitosti $\mathcal{O}(N^2)$. Tedy zanedbáváme multiplikativní konstantu, aditivní konstantu a nižší řády funkce, protože nás zajímá, jak se bude měnit časová složitost vzhledem ke zvětšujícím se vstupním datům.

Tím, že zanedbáváme multiplikativní a aditivní konstantu a nižší řády funkce, může se vyskytnout případ, kdy algoritmus s vyšším řádem asymptotické složitosti proběhne rychleji než algoritmus s nižším řádem. Máme ovšem jistotu, že existuje vstup o velikosti N_0, kde toto přestane platit. Při praktickém použití tato situace

může nastat v případě, kdy asymptoticky rychlejší algoritmus má složitější implementaci a prostředky vynaložené na režii převyšují výhody algoritmu. Takováto situace je vyjádřena na obrázku 4.1, kde vidíme, že díky zanedbaným multiplikativním a aditivním konstantám může být reálný čas výpočtu asymptoticky rychlejšího algoritmu menší než u algoritmu s vyšší asymptotickou časovou složitostí. Je zde ovšem vidět, že při velikosti vstupu N_0 se situace obrátí a dále se chovají tak, jak bychom očekávali [16].



Obrázek 4.1: Graf znázorňující časový průběh algoritmu v závislosti na velikosti vstupu N. (zdroj: autor)

Pro vyjádření časové složitosti můžeme využít různé varianty.

- Horní odhad složitosti - $\mathcal{O}(f(N))$
- Průměrná složitost - $\Theta(f(N))$
- Dolní odhad složitosti - $\Omega(f(N))$

Často vídané vyjádření složitosti je takzvaná *Omikron notace*, která vyjadřuje horní odhad, tedy nejhorší možný případ, jakým se může algoritmus chovat. V tomto případě máme jistotu, že algoritmus bude asymptoticky probíhat stejně rychle nebo rychleji v případě časové složitosti. V některých případech se vyplatí uvádět i průměrnou složitost, tedy složitost, která nastává při náhodném rozložení vstupních dat. Typickou ukázkou, kde je vhodné využít průměrnou časovou složitost, je metoda *Quick-Sort*, která patří mezi třídící algoritmy. Horní odhad složitosti je

N	10	20	40	60	500	1000
$\log n$	2,3µs	4,3µs	5µs	5,8µs	9µs	10µs
n	10µs	20µs	40µs	60µs	0,5s	1ms
$n \log n$	23µs	86µs	0,2ms	0,35ms	4,5ms	10ms
n^2	0,1ms	0,4ms	1,6ms	3,6ms	0,25s	1s
n^3	1ms	8ms	64ms	0,2s	125s	17min
n^4	10ms	160ms	2,56s	13s	17h	11,6dní
2^n	1ms	1s	12,7 dní	36000 let		
$n!$	3,6s	77000 let				

Tab. 4.1: Příklad závislosti výpočetního času na asymptotické složitosti algoritmu. (zdroj: autor)

$\mathcal{O}(n^2)$. Při vhodné implementaci se ovšem lze kvadratické složitosti účinně bránit a obecně se algoritmus chová s průměrnou časovou složitostí $\Theta(n \log n)$, což je pro třídící algoritmy nejlepší možné [30]. Poslední variantou je dolní odhad složitosti, který nám naopak vyjadřuje, že algoritmus se bude chovat stejně nebo hůře než je uvedeno. Toto vyjádření využijeme především v oblasti kryptografie [10, 20].

4.1.2 Stanovení časové složitosti

Pro určení časové složitosti je nutné určit počet operací v závislosti na velikosti vstupních dat. Nezajímáme se o přesný počet operací, ale o vztah počtu operací k velikosti vstupní množiny. Pro komplikované algoritmy to může být nesnadný úkol. U geometrických algoritmů se často zajímáme pouze o horní odhad složitosti díky jeho relativně jednoduchému zjištění oproti průměrnému odhadu.

4.2 Metody algoritmizace ve výpočetní geometrii

Abychom byli schopni porozumět různým algoritmům nebo navrhnout nový algoritmus pro různé operace ve výpočetní geometrii, je dobré se seznámit s často používanými metodami v algoritmech výpočetní geometrie. Metody návrhů alogritmů neřeší konkrétní problém, ale pouze obecný postup, jak se při hledání řešení chovat.

4.2.1 Metoda hrubé síly

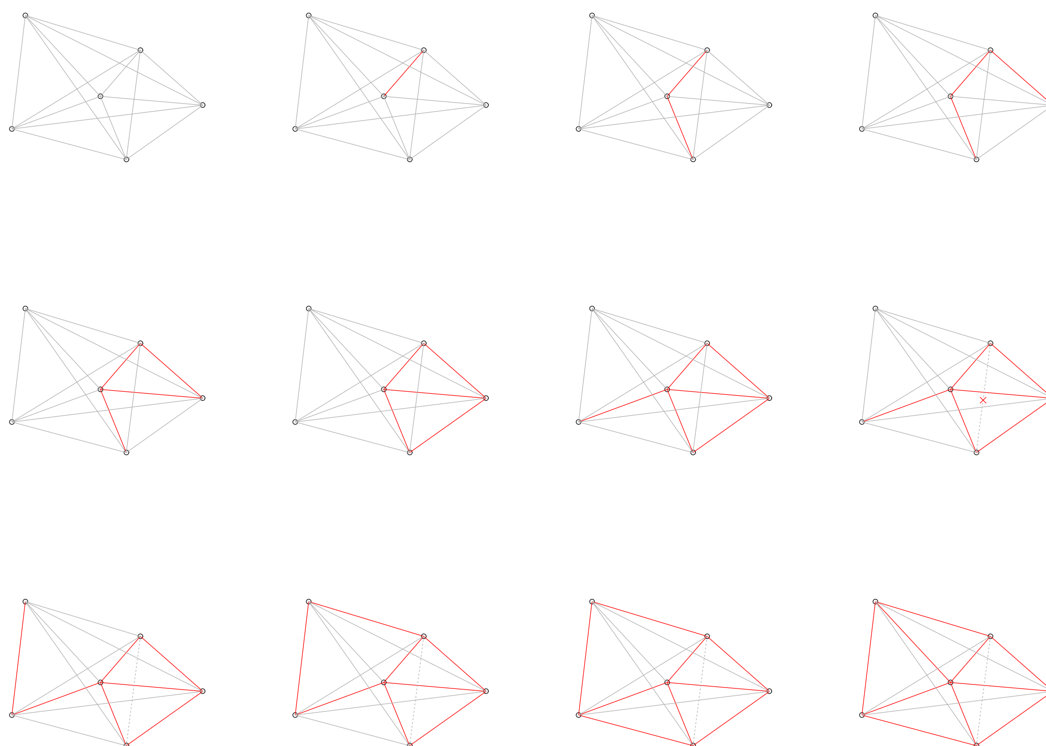
Patří k nejjednodušším metodám algoritmizace. Jak vyplývá z názvu, v této metodě budeme postupovat hrubou silou, tedy v praxi to znamená, že algoritmus bude zkoušet všechny možné kombinace řešení. Mezi výhody patří jednoduchá implementace a snadné porozumění algoritmu, které se často shoduje s definicí výsledku. Tyto algoritmy lze využívat zpravidla pro malé vstupní množiny nebo pro ověření správnosti výsledku. V praxi lze algoritmy využít i jako doplněk složitějších algoritmů, kde by režie, jako je volání funkcí nebo vytváření objektů, zabrala více času než na tuto malou množinu použít algoritmus hrubé síly. Velmi často je časová složitost těchto algoritmů $\mathcal{O}(2^n)$ či dokonce $\mathcal{O}(n!)$. V tabulce 4.1 pak můžeme vidět, že časová složitost pro větší vstupy je neúnosná.

4.2.2 Heuristické algoritmy

Heuristické algoritmy se velmi podobají metodě hrubé síly, pouze nezkoumají všechna možná řešení. Tato technika je velmi často používaná u optimalizačních úloh, kde neexistuje exaktní řešení. Zatímco u algoritmu řešícího úlohu metodou hrubé síly bylo nalezení optimálního řešení zaručeno, heuristické algoritmy hledají pouze přípustné řešení. Tímto způsobem docílíme lepší časové náročnosti oproti algoritmům hrubé síly. Jednou ze známých metod heuristických algoritmů jsou takzvané hladové algoritmy. Ty se nalezením lokálně optimálních řešení snaží docílit globálního optima. Příkladem hladového algoritmu může být nalezení nepravidelné trojúhelníkové sítě nad množinou bodů v rovině, kdy postupně přidáváme do řešení nejkratší hrany, pokud ovšem nekříží jinou hranu vyskytující se již v řešení.

4.2.3 Rozděl a panuj

Nejen ve výpočetní geometrii je známá a hojně používaná metoda *rozděl a panuj*, známá také pod anglickým názvem *divide and conquer*. Paradigma je založeno na rekurzivním rozdělování problému na dvě či více částí stejného nebo podobného typu, dokud nebudeme schopni problém jednoduše vyřešit. Řešení dílčích problémů se pak spojuje a získá se řešení původního problému. Metoda *rozděl a panuj* je vhodná pro paralelní zpracování, kde při každém rozdělení nám vznikají dvě či více částí, které můžeme řešit nezávisle na sobě [15]. Do algoritmů této kategorie patří například generalizační algoritmus *Douglas-Peucker*[27], který je vhodný svou

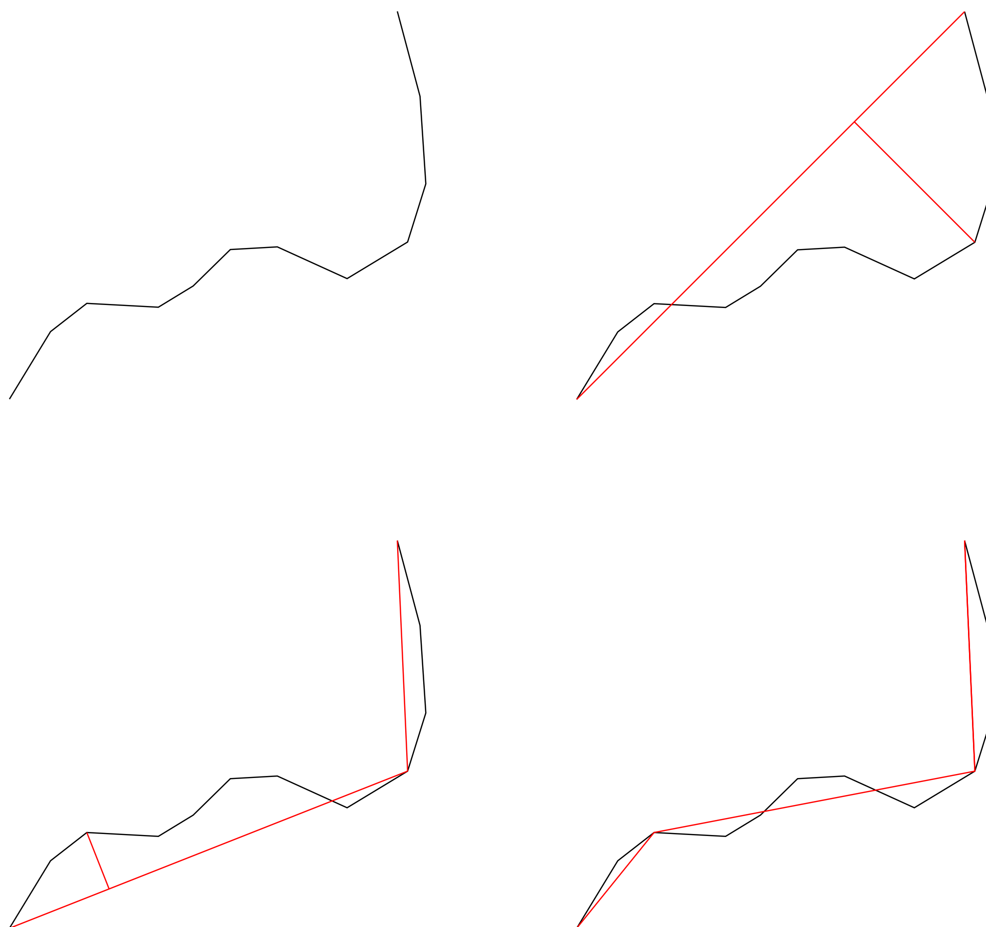


Obrázek 4.2: Znázornění postupu algoritmu pro nalezení triangulace metodou hladového algoritmu. (zdroj: autor)

jednoduchostí na ukázku metody rozděl a panuj. Na vstupu dostává algoritmus linii, kterou chceme generalizovat, a hodnotu vzdálenosti, o kterou se generalizovaná linie může maximálně lišit od původní. Funkce vyhledá bod s maximální vzdáleností od úsečky spojující počáteční a koncový bod, který je zároveň vzdálenější než námi zadaná mez, poté se rekurzivně zavolá na dvě úsečky obsahující tento bod. Takto se postupuje do té doby, dokud existuje bod s větší vzdáleností než zadaná mez. Po dokončení se řešení složí z jednotlivých kroků rekurze.

4.2.4 Zametací přímka

Další metoda využívaná výhradně ve výpočetní geometrii je metoda takzvané *zametací přímky* neboli *sweep line*. Myšlenkou této techniky je procházení setříděných dat. Ve výpočetní geometrii se nejčastěji jedná o průchod bodů seřazených podle jedné ze souřadnic. Pokud takto setříděnou vstupní množinou nedisponujeme, je nutno nejprve data setřídit jedním ze známých třídících algoritmů. Pak lze průchod



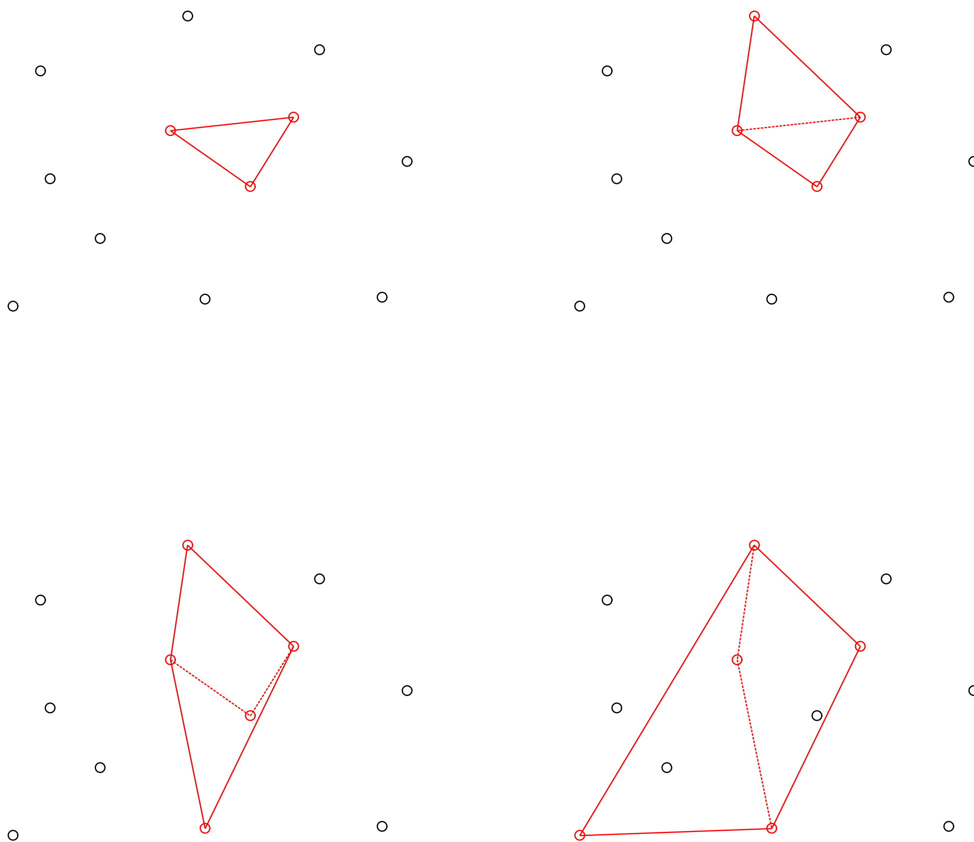
Obrázek 4.3: Znázornění postupu *Douglas-Peucker* algoritmu pro ukázku metody *rozděl a panuj*. (zdroj: autor)

body vizualizovat podle implementace, například jako svislou přímku pohybující se zleva doprava po množině bodů. Tuto metodu využívá například *Bentley-Ottmannův* algoritmus pro nalezení průsečíků množin linií, který je podrobněji popsán v kapitole 5, proto se jím zde nebudeme zabývat.

4.2.5 Inkrementální algoritmy

Algoritmy tohoto typu se snaží dosáhnout výsledku tím způsobem, že ze vstupní množiny přidávají nebo aktualizují prvky ve výstupní množině. Tedy po každém kroku je řešení aktualizováno, dokud není dosaženo konečného výsledku. Výhodou této metody je, že ji často lze použít pro online algoritmy, tedy do vstupní množiny

mohou být přidávány prvky v průběhu výpočtu. Pro ukázku zde byl vybrán algoritmus pro inkrementální výpočet konvexní obálky. Algoritmus vybere bod ze vstupní množiny a určí polohu vůči aktuálním liniím konvexní obálky. Pokud bod vůči nějakým hranám leží v pravé polorovině, tyto hrany jsou označeny čárkovaně, algoritmus odebere tyto hrany z konvexní obálky a mezi volné konce přidá aktuálně zkoumaný bod. Pokud bod leží vůči všem hranám v levé polorovině, řešení se neaktualizuje a pokračuje se dál. Na tomto příkladu je vidět i možnost využívat algoritmus online, tedy kdykoliv do výpočtu zahrnout další body.



Obrázek 4.4: Znázornění postupu inkrementálního algoritmu pro konvexní obálku. (zdroj: autor)

5 Rešerše používaných algoritmů

Tato kapitola se zabývá přehledem využívaných algoritmů pro tvorbu polygonů z množiny linií. Polygonizace, tak jak jí můžeme pozorovat v různých GIS softwarech, se zpravidla neřeší jedním algoritmem, ale používají se různé algoritmy pro jednotlivé kroky polygonizace. Popíšeme zde tedy do jakých kroků lze polygonizaci rozdělit a jak jednotlivé kroky optimálně řešit.

Polygonizaci lze nejjednodušeji rozdělit na 2 základní kroky. Jako vstupní data budeme uvažovat množinu linií, tyto linie ovšem nemusí obsahovat, v reálném použití často neobsahují, body ve společných průsečících. Tato situace nastává především v případech, používáme-li různé vrstvy pro tvorbu polygonů. V terminologii GIS se tyto průsečíky označují jako *fuzzy*, volně přeloženo jako *neostré*. Abychom mohli zahájit další krok polygonizace, je nutné tyto průsečíky doplnit. V první části se tedy budeme zabývat, jak optimálně doplnit průsečíky množin linií.

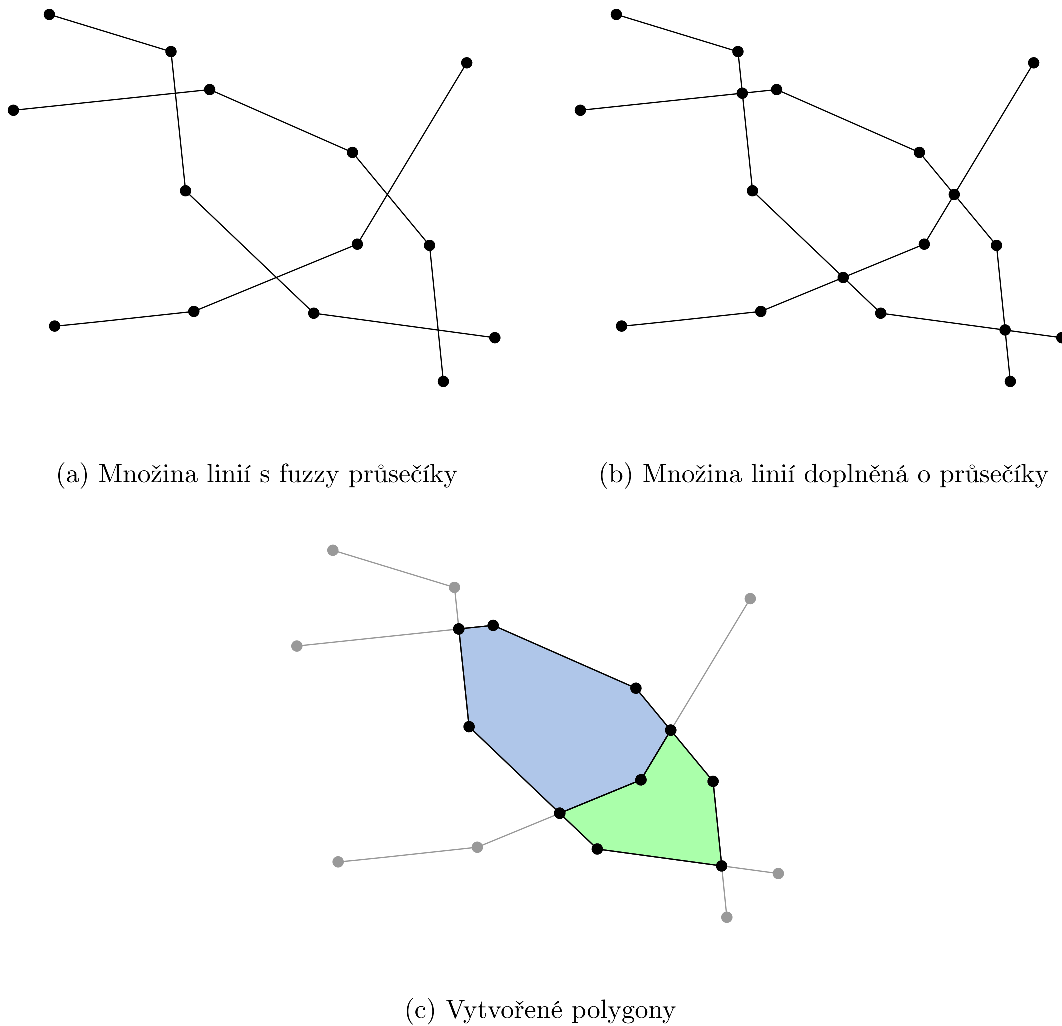
Předpokládejme, že množina linií je zbavená *fuzzy* průsečíků, tedy každá dvojice úseček má společný nanejvýš jeden koncový bod. V takovéto situaci můžeme přistoupit k dalšímu kroku, tedy vlastní tvorbě polygonů z množiny linií. Polygony, které hledáme, musí splňovat podmínku, že žádný námi vytvořený polygon nemůže být rozdělen jakoukoliv linií na více menších polygonů. Jelikož se v podstatě jedná o určité vyjádření grafu, pro tvorbu polygonů se využívá grafových algoritmů.

5.1 Výpočet průsečíků množiny linií

Výpočet všech průsečíků množin linií lze provést snadno, testováním všech úseček se všemi. Tímto postupem ovšem zjevně dosáhneme složitosti $\mathcal{O}(N^2)$, kde N je počet segmentů linie. Pro výpočet průsečíků lze ovšem využít i algoritmus s časovou náročnosti $\mathcal{O}(n \log n)$, známý také jako Bentley–Ottmannův algoritmus.

5.1.1 Vzájemná poloha dvou úseček

Ve 2D výpočetní geometrii jsou standardně jednotlivé segmenty linie vyjádřeny počátečními a koncovými souřadnicemi. Uvažujme tedy, že máme dány 2 přímky $p_1 = |S_1E_1|$ a $p_2 = |S_2E_2|$, kde $S_1 = [x_{S1}, y_{S1}]$, $E_1 = [x_{E1}, y_{E1}]$, $S_2 = [x_{S2}, y_{S2}]$, $E_2 = [x_{E2}, y_{E2}]$, a potřebujeme provést test, zda se dané přímky protínají či nikoliv. Můžeme použít takzvaný *Half-Plane* test, tedy test, který určuje, zda bod leží v pravé či levé polorovině od přímky. Tento test zopakujeme celkem čtyřikrát a to na



(a) Množina linií s fuzzy průsečíky

(b) Množina linií doplněná o průsečíky

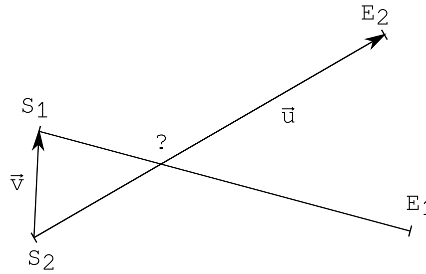
(c) Vytvořené polygony

Obrázek 5.1: Znázornění postupu polygonizace. (zdroj: autor)

počáteční a koncový bod druhé úsečky, abychom zjistili, zda se přímky protínají. Test je založen na výpočtu orientace dvou vektorů $\vec{u}$ a $\vec{v}$, kde vektor $\vec{u}$ je směrový vektor úsečky $\overrightarrow{S_1E_1}$ a vektor $\vec{v}$ je vektor $\overrightarrow{S_1S_2}$. Po aplikaci výpočtu orientace vektorů na každý z koncových bodů jsme schopni jednoduše určit, zda se dané přímky protínají. Tento postup nám ovšem nedá souřadnice samotného průsečíku.

5.1.2 Nalezení průsečíku dvou úseček

Pro nalezení průsečíku P dvou úseček vycházíme z parametrického vyjádření přímek, jelikož obecné rovnice neumožňují pracovat s přímkami rovnoběžnými s osou y. Uvažujme tedy stejné značení bodů jako v předchozím odstavci. Nejprve je tedy nutné vyjádřit rovnici přímky z počátečního a koncového vrcholu. Pro parametrické



Obrázek 5.2: *Half-plane* test. (zdroj: autor)

vyjádření využijeme směrový vektor přímky a libovolný bod na přímce. Ovšem pro zjednodušení výpočtů použijeme pro přímku p_1 směrový vektor $\overrightarrow{S_1E_1}$ a jako bod do parametrické rovnice dosadíme S_1, obdobně zkonstruujeme i parametrické vyjádření přímky p_2. Soustava rovnic pro výpočet průsečíku p_i pak tedy bude vypadat takto:

$$P = S_1 + s \cdot \overrightarrow{S_1E_1},$$

$$P = S_2 + t \cdot \overrightarrow{S_2E_2}.$$

Rozepsáno podle souřadnic:

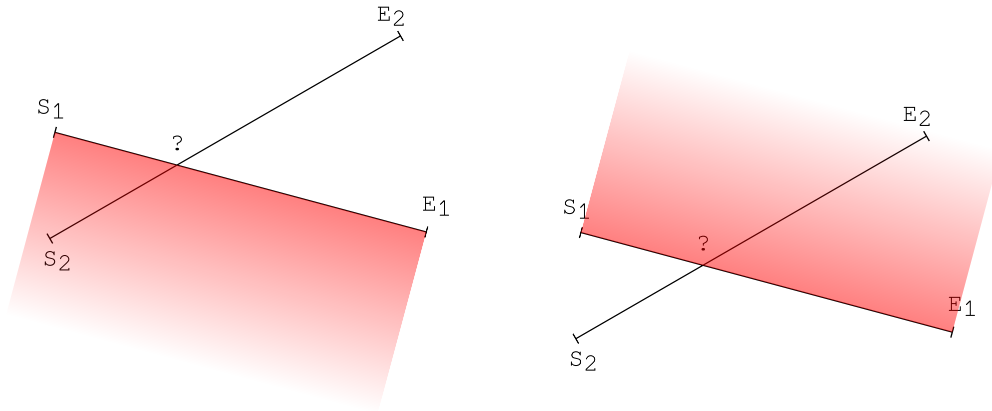
$$x_P = x_{S1} + s(x_{E1} - x_{S1}),$$

$$y_P = y_{S1} + s(y_{E1} - y_{S1}),$$

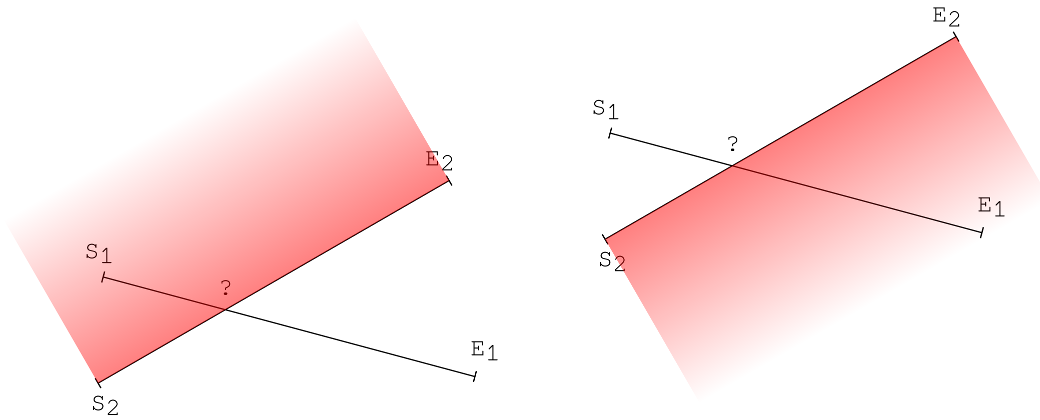
$$x_P = x_{S2} + t(x_{E2} - x_{S2}),$$

$$y_P = y_{S2} + t(y_{E2} - y_{S2}).$$

Pro získání průsečíku musíme nejprve ze soustavy rovnic vyjádřit parametry s a t. To provedeme následujícím způsobem:



(a) Test bodu S_2 k přímce $|S_1E_1|$. (zdroj: autor) (b) Test bodu E_2 k přímce $|S_1E_1|$. (zdroj: autor)



(c) Test bodu S_1 k přímce $|S_2E_2|$. (zdroj: autor) (d) Test bodu E_1 k přímce $|S_2E_2|$. (zdroj: autor)

Obrázek 5.3: Grafické znázornění *Half-plane* testů pro zjištění existence průsečíku. (zdroj: autor)

$$s = \frac{y_{S1}(x_{S2} - x_{E2}) + y_{S2}(x_{E2} - x_{S1}) + y_{E2}(x_{S1} - x_{S2})}{(x_{E1} - x_{S1})(y_{S2} - y_{E2}) - (y_{E1} - y_{S1})(x_{S2} - x_{E2})},$$

$$t = \frac{y_{S1}(x_{S2} - x_{E1}) + y_{E1}(x_{S1} - x_{S2}) + y_{S2}(x_{E1} - x_{S1})}{(x_{E1} - x_{S1})(y_{S2} - y_{E2}) - (y_{E1} - y_{S1})(x_{S2} - x_{E2})}.$$

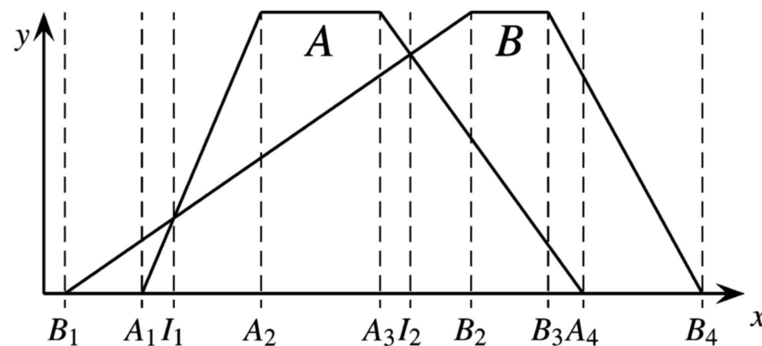
Z charakteru parametrického vyjádření přímek poté můžeme rozhodnout o vzájemné poloze úseček. Pokud jsou parametry $s \in < 0, 1 >, t \in < 0, 1 >$, pak se úsečky protínají, v opačném případě se úsečky neprotínají. Je potřeba ošetřit případy, kdy úsečky leží na jedné přímce. Podrobněji je tento způsob popsán v publikaci [10].

5.1.3 Bentley–Ottmannův algoritmus

Tento algoritmus je založen na technice zvané *sweep line* neboli *zametací přímka*. Tato technika je již zmíněna v kapitole 4. Hlavní myšlenka pro zrychlení algoritmu oproti metodě hrubé síly je počítat průsečíky pouze sousedních linií, tedy linií, které jsou právě protnuty zametací přímkou. Tímto způsobem se vyhneme časové složitosti $\mathcal{O}(N^2)$ a docílíme mnohem lepšího výsledku, tedy složitosti $\mathcal{O}(n \log n)$ [11].

Prvním a zásadním krokem algoritmů využívajících techniku *zametací přímky* je setřídit vstupní data podle jedné ze souřadnic. Setřídíme tedy jednotlivé souřadnice podle jedné z nich, často se používá souřadnice x, pak můžeme vizualizovat *zametací přímku* jako svislou linii pohybující se zleva doprava. Toto setřídění lze provést jedním ze známých třídících algoritmů v čase $\mathcal{O}(n \log n)$.

Setříděné souřadnice jsou pak vloženy do fronty, odkud jsou postupně zpracovávány a hledány průsečíky jen těch linií, které jsou aktuálně protnuty zametací přímkou, dokud algoritmus nenalezne veškeré průsečíky. Podrobnější popis algoritmu najdeme v publikacích [11, 13].



Obrázek 5.4: Znázornění postupu zametací přímky v Bentley-Ottmanově algoritmu. Převzato z [13].

5.2 Tvorba polygonů z množiny linií

Předpokládejme, že máme množinu linií doplněnou o průsečíky, tedy každá dvojice úseček v této množině sdílí nejvýše koncový bod. V předchozí sekci bylo řečeno, že průsečíky linií lze doplnit pomocí *Bentley–Ottmannova* algoritmu v čase $\mathcal{O}(n \log n)$. Nyní chceme tedy v této množině linií nalézt všechny polygony, tak aby jednotlivé polygony spolu sdílely maximálně hrany a zároveň aby žádný z polygonů neobsahoval

jiný polygon. Takové linie jsou v podstatě reprezentací grafu. K nalezení polygonů můžeme použít tedy grafových algoritmů.

Tak jak je polygonizace popsána v publikaci [18], je prvním krokem k nalezení polygonů **nalezení nejkratších cest v grafu**. Pro nalezení nejkratších cest nám může posloužit jeden ze známých algoritmů. Nalezení nejkratších cest mezi všemi dvojicemi vrcholů v grafu jsme schopni realizovat v čase $\mathcal{O}(n^3)$, kde n je počet vrcholů. Můžeme využít například

- Floydův–Warshallův algoritmus,
- Dijkstrův algoritmus,
- Johnsonův algoritmus.

Jako druhý krok lze považovat **nalezení nejkratších cyklů**, tedy v podstatě vlastních polygonů. V publikaci [18] je pro nalezení nejkratších cyklů použitý jednoduchý hladový algoritmus.

6 Konvenční prostředky a polygonizace

Existuje celá řada konvenčních nástrojů, kterými lze řešit polygonizaci. Oblast GIS je známá značným množstvím kvalitních nástrojů s otevřeným zdrojovým kódem. Můžeme tak nalédnout do implementace jednotlivých softwarových řešení. Výhodou je i velká komunita, která vždy ochotně poradí. Mimo jiné jsou zde i proprietární zástupci, kteří nám možnost nahlédnout do zdrojového kódu zpravidla nenabídnou. Zato by nám měli poskytovat uživatelskou podporu, za kterou zákazník samozřejmě platí koupí softwaru.

Mohlo by se zdát, že nástrojů existuje celá řada, tak proč vytvářet nástroj nový? Jedná se zde především o spolehlivost a jednoduchost. Jak u komerčních, tak u open source nástrojů je s nadsázkou téměř tradicí, že po přechodu na vyšší verzi nějaké komponenty přestanou fungovat a končí na neočekávaných chybách. Výjimkou nejsou ani nástroje pro polygonizaci.

Zmíníme zde tedy nejvýznamnější zástupce z oblasti GIS a ukážeme, které nástroje z těchto softwarů lze pro polygonizaci využít.

6.1 ArcGIS Desktop

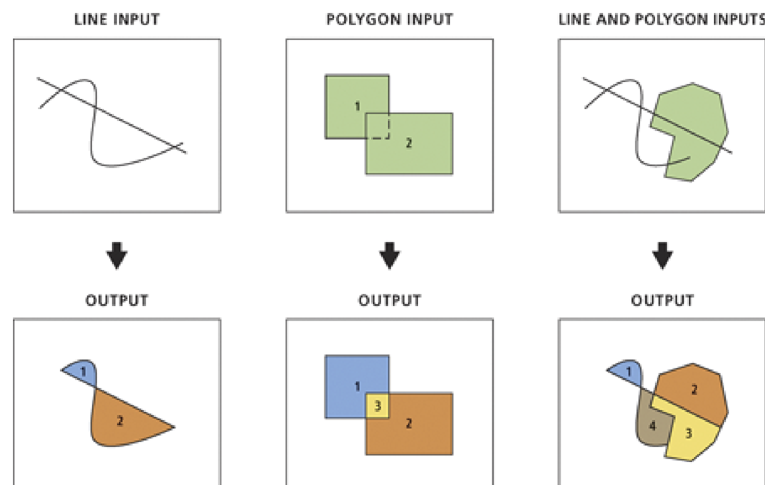
ArcGIS je vyvíjený společností Esri, v současné době se jedná o jeden z nejvyužívanějších nástrojů v oblasti GIS. Jedná se o placený software, za který jednotlivec v současné chvíli zaplatí 100 dolarů ročně [5]. Používat tento nástroj tedy pouze pro polygonizaci by bylo nesmyslné, ovšem pokud uživatel již tímto softwarem disponuje, je toto jedna z možností.

Feature to Polygon

Feature to Polygon je nástroj sloužící pro tvorbu polygonů. Jako vstupní data mohou být použity linie i polygony. Výstupní polygony z tohoto nástroje na testovacích datech byli korektní.

6.2 QGIS Desktop

Přímým konkurentem softwaru ArcGIS Desktop je bezpochyby vydařený QGIS Desktop. Jedná se o nástroj vyvíjený komunitou pod záštitou OSGeo. Je šířen pod copyleftovou licencí *GNU General Public License*, tudíž máme volný přístup ke zdrojovému kódu aplikace dostupnému v online repozitářích. To nám umožňuje nahlížet



Obrázek 6.1: Ilustrace vstupu a výstupu nástroje Feature to Polygon. Převzato z [4].

do výpočetních algoritmů, které jsou v případě QGIS psány v programovacím jazyce *C++* a *Python*, na rozdíl od komerčních nástrojů, které si implementaci často chrání. Výstup z nástroje byl opět validní. Na rozdíl od nástroje z ArcGIS Desktop se ovšem podařilo zpracovat i větší množství dat.

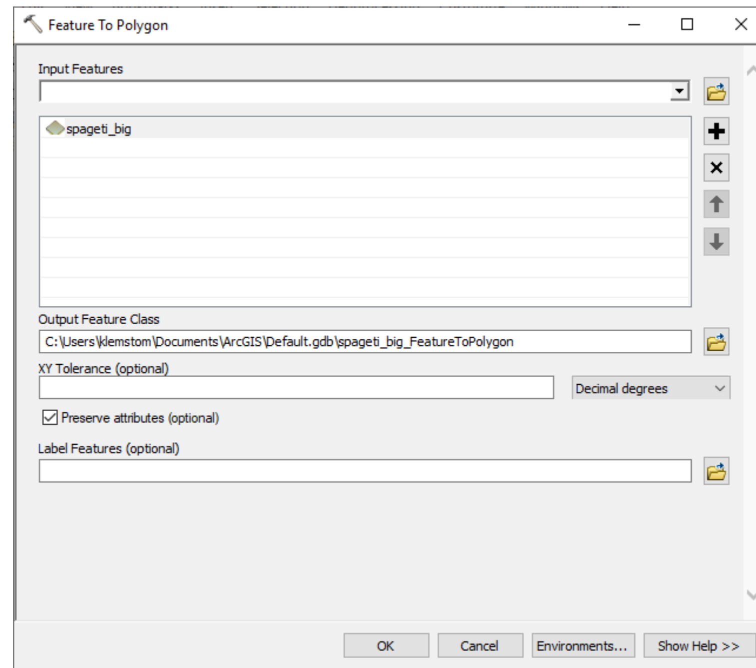
Polygonize

Nástroj Polygonize je od verze 3.12 napsán v *C++*, v předchozích verzích QGIS Desktop byl zakomponován formou pluginu psaného v jazyce Python. Využívá metod knihovny GEOS. Důležitý rozdíl oproti nástroji z ArcGIS Desktop je, že na vstupu neakceptuje jiné prvky než linie [23].

6.3 PostGIS

Obdobně byl testován i nástroj PostGIS. Zde už se nejedná o desktopovou aplikaci, jako tomu bylo u výše testovaných softwarů. PostGIS je nadstavba pro objektově-relační databázi PostgreSQL. Jedná se tedy o rozšíření databáze o podporu geografických objektů a funkcí. Je často využíván k uchování geografických dat a analýz nad nimi. Nutno podotknout, že PostGIS využívá pro většinu svých operací taktéž knihovnu *GEOS* psanou v programovacím jazyce *C++*, kterou následně poskytuje uživateli přes standardní dotazovací jazyk SQL.

Postup polygonizace zde bude o něco složitější. Budeme nuceni polygonizaci rozdělit do dvou kroků, jak bylo řečeno v kapitole 5. Nejprve se tedy zaměříme



Obrázek 6.2: Rozhraní nástroje Feature to Polygon v softwaru ArcGIS Desktop. (zdroj: autor)

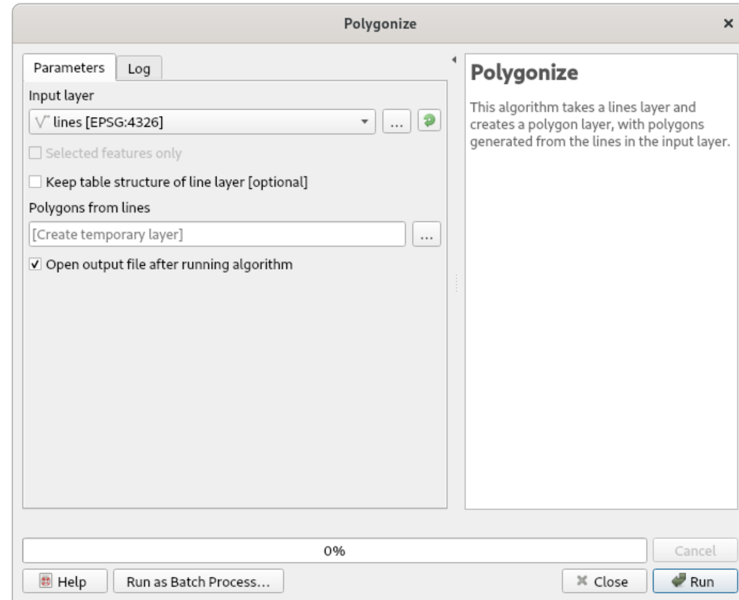
na doplnění průsečíků linií a poté provedeme vlastní polygonizaci. Pro doplnění průsečíku linií nám PostGIS nabízí hned dvě možnosti.

Funkce *ST_UnaryUnion*

První z možností je využít funkci *ST_UnaryUnion*. Tato funkce rozpouští linie tak, že sdílí maximálně koncový bod s jinými liniemi. Tímto způsobem můžeme připravit podkladové linie pro následnou polygonizaci. Nevýhoda této funkce je ovšem v její časové složitosti. Oproti druhé možnosti, tedy funkci *ST_Node*, dosahuje značně vyšších časů pro výpočet [22].

Funkce *ST_Node*

Druhá možnost a alternativa funkce *ST_UnaryUnion* je funkce *ST_Node*. Tato funkce taktéž rozpojuje linie v průsečících. Nicméně oproti předchozí zmíněné pracuje v mnohem kratších časech. Výhodou obou těchto funkcí může být možnost pracovat s 3D daty [22].



Obrázek 6.3: Rozhraní nástroje Polygonize v softwaru QGIS Desktop. (zdroj: autor)

```
SELECT st_polygonize(lines_nodded.geom)
FROM (
        SELECT st_node(st_collect(st_linemerge(geom))) AS geom
        FROM lines
    ) AS lines_nodded
```

Obrázek 6.4: Příklad tvorby polygonů v PostGIS. (zdroj: autor)

Funkce *ST_Polygonize*

Pokud máme průsečíky doplněny, můžeme přistoupit k vlastní tvorbě polygonů. K tomu nám poslouží funkce *ST_Polygonize* využívající metod knihovny GEOS. Výstupem této funkce je pak geometrická kolekce obsahující polygony [22].

7 Návrh a implementace

Pro implementaci polygonizačního algoritmu byla vybrána open-source knihovna pro programovací jazyk Java *GeoTools*. V současné době je *GeoTools* součástí *OSGeo*, což je nezisková organizace podporující open-source geoprostorové technologie. Mimo *GeoTools* organizace zastupuje řadu dalších projektů, mezi které patří v desktopových aplikacích textitQGIS, *GRASS GIS*, mezi mapové servery *MapServer*, *GeoServer* a mezi knihovnami jsou nejvýznamnějšími zástupci *GEOS*, *GDAL*, *PROJ*, nebo *JTS* [3]. Právě zmíněná knihovna *JTS* tvoří základní kámen pro *GeoTools*, jelikož jsou v ní definovány základní geometrické typy a operace pro práci s prostorovými daty [1]. Možná více známá knihovna než *JTS* je knihovna *GEOS*, což je přepis knihovny z Javy do C++ [1, 3].

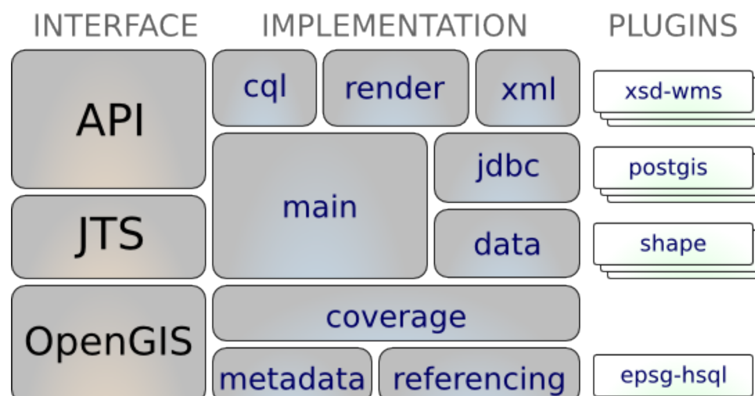
7.1 GeoTools

GeoTools je toolkit pro práci s prostorovými daty, je šířen pod licencí LGPL, která nám umožňuje knihovnu využívat i v proprietárním softwaru, avšak má své omezení pokud bychom měli v plánu vytvořit odvozené dílo [6]. Datové struktury jsou založeny na specifikacích OGC, která poskytují standardy pro geo prostorová data. Na obrázku 7.2 jsou znázorněny komponenty, ze kterých se geotools skládá. Je zde například vidět, že GeoTools poskytuje sadu JTS. GeoTools využívá software jako například GeoServer, uDig, Geopublisher, Geomajas a mnoha dalších [1].

Obrázek 7.1: Logo GeoTools. Převzato z [1].

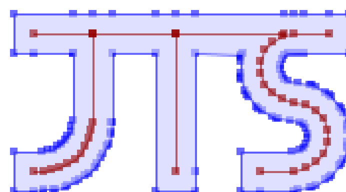
7.2 JTS Topology Suite

JTS je knihovna pro vytváření a manipulaci s vektorovou geometrií psaná v Javě. Poskytuje nám řadu nástrojů pro práci s vektorovými daty, které budeme využívat při implementaci polygonizačního algoritmu, neimplementuje však například formáty pro uložení dat. JTS je šířena pod licencí BSD [7], která nám umožňuje



Obrázek 7.2: Struktura GeoTools. Převzato z [1].

knihovnu používat v podstatě bez omezení. Jedná se o jednu z nejsvobodnějších licencí [6].



Obrázek 7.3: Logo JTS Topology Suite. Převzato z [7].

7.3 Návrh

Polygonizační proces byl navrhnut tak, aby bylo maximálně využito funkcionality JTS Topology Suite. Jelikož tato knihovna poskytuje základní sadu algoritmů pro práci s prostorovými daty, bylo využito těchto již naimplementovaných a otestovaných algoritmů.

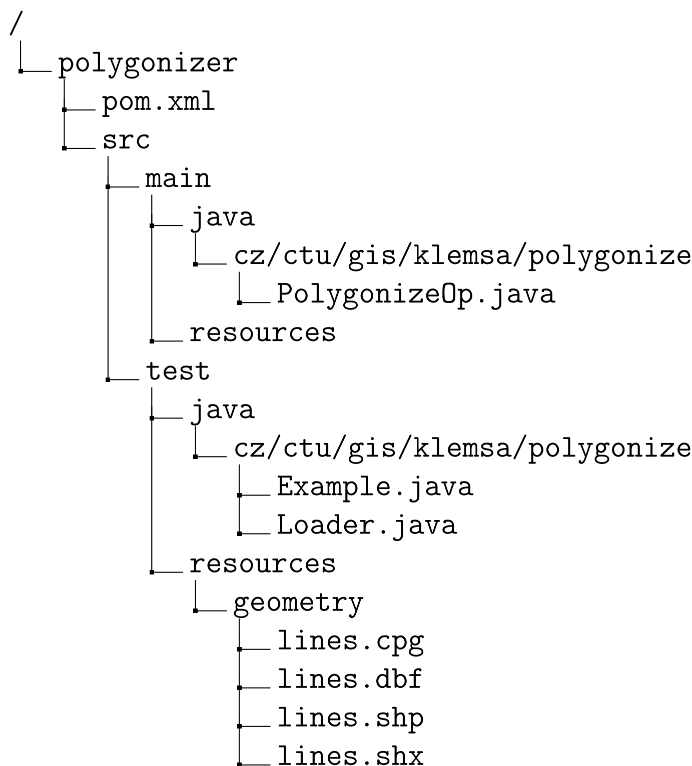
Třída UnaryUnionOp sjednocuje kolekci geometrií nebo samotnou geometrii. U linií má za následek rozpojení linií v průsečících. Tato třída byla tedy použita pro přípravu linií pro další krok.

Třída Polygonizer vytváří polygony z linií, které reprezentují rovinný graf. Rovinný graf jsme vytvořili v předchozím kroku rozpojením linií v průsečících.

7.4 Implementace

Jako nástroj pro správu buildů byl použit nástroj Apache Maven, který je ve velké míře používán právě pro Java projekty, ovšem nic nebrání použití na různé programovací jazyky. Typickým znakem pro maven je konfigurační soubor *pom.xml*, který nám slouží pro definici závislostí a mnoha dalších parametrů. Pro sestavení je potřeba mít v počítači nainstalovaný nástroj Maven [8]. Poté v adresáři se souborem *pom.xml* příkazem *mvn package* jednoduše sestavíme projekt a Maven nám vytvoří **.jar* archiv. Výhodou tohoto nástroje je, že s kódem nemusíme distribuovat knihovny, které využívá. Maven sám zajistí jejich stažení z repozitářů a poskytne je.

Hlavním prvkem projektu je třída *PolygonizeOp*, která implementuje operaci polygonizace. Na obrázku 7.4 je pak vidět celá adresářová struktura projektu. Třída *PolygonizeOp* uživateli nabízí metody pro polygonizaci.



Obrázek 7.4: Struktura projektu. (zdroj: autor)

7.4.1 Třída PolygonizeOp

PolygonizeOp je třída operace polygonizace implementovaná za pomocí metod JTS Topology Suite. Celou implementaci třídy nalezneme v příloze B.

Práce se třídou

Třída poskytuje dva veřejné konstruktory, kterým lze přidat linie do výpočtu.

```
public PolygonizeOp(Collection geoms)
public PolygonizeOp(Collection geoms, GeometryFactory geomFact)
```

Výstupní polygony získáme pak metodou *polygonize()*.

```
public Collection polygonize()
```

Kromě přístupu přes instanci, třída také nabízí přístup přes statické metody.

```
public static Collection polygonize(Collection geoms)
public static Collection polygonize(Collection geoms,
↪  GeometryFactory geomFact)
public static Collection polygonize(Geometry geom)
```

7.5 Příklad použití

Projekt obsahuje třídu *Example*, která na ukázku načítá linie z shapefilu, provede polygonizaci a výsledné polygony vypíše ve formě WKT na standardní výstup. Při větších objemech dat může nastat komplikace s nedostatkem paměti, který operační systém poskytuje Java Virtual Machine. proto je potřeba tyto limity v některých případech zvýšit.

8 Závěr

V této práci byla zpracována rešerše používaných algoritmů pro polygonizaci, kde bylo zároveň naznačeno, jak polygonizaci řeší nejrůznější software. Bylo zjištěno, že ve většině případů polygonizace probíhá v prvním kroku doplněním průsečíků linií a vytvořením reprezentace rovinného grafu, následně se postupuje za pomoci grafových algoritmů k nalezení polygonů. Toto řešení jsme mohli pozorovat například u aplikace QGIS Desktop.

Hlavním cílem této práce bylo navrhnout a implementovat algoritmus pro polygonizaci, který bude vhodný pro automatizaci procesu vytváření pátracích sektorů. Pro implementaci bylo využito knihovny JTS Topology Suite, která nám poskytla základní objekty a operace, nichž bylo při implementaci využito. Pro testování implementovaného algoritmu byla využita knihovna GeoTools.

Seznam zkratek

GIS	Geografický informační systém
OSGeo	Open Source Geospatial Foundation
GEOS	Geometry Engine, Open Source
OGC	Open Geospatial Consortium
GPL	Všeobecná veřejná licence (General Public License)
LGPL	Lesser General Public License
WKT	Well-known text
RGB	Barevný model Red, Green, Blue
GDF/DIME	Geographic Base File/Dual Independent Map Encoding

Seznam obrázků

3.1 Ukázka topologického modelu (převzato z [17]) 14
4.1 Graf znázorňující časový průběh algoritmu v závislosti na velikosti vstupu N. (zdroj: autor) 16
4.2 Znázornění postupu algoritmu pro nalezení triangulace metodou hladového algoritmu. (zdroj: autor) 19
4.3 Znázornění postupu *Douglas-Peucker* algoritmu pro ukázku metody *rozděl a panuj*. (zdroj: autor) 20
4.4 Znázornění postupu inkrementálního algoritmu pro konvexní obálku. (zdroj: autor) 21
5.1 Znázornění postupu polygonizace. (zdroj: autor) 23
5.2 *Half-plane* test. (zdroj: autor) 24
5.3 Grafické znázornění *Half-plane* testů pro zjištění existence průsečíku. (zdroj: autor) 25
5.4 Znázornění postupu zametací přímky v Bentley-Ottmanově algoritmu. Převzato z [13]. 26
6.1 Ilustrace vstupu a výstupu nástroje Feature to Polygon. Převzato z [4]. 29
6.2 Rozhraní nástroje Feature to Polygon v softwaru ArcGIS Desktop. (zdroj: autor) 30
6.3 Rozhraní nástroje Polygonize v softwaru QGIS Desktop. (zdroj: autor) 31
6.4 Příklad tvorby polygonů v PostGIS. (zdroj: autor) 31
7.1 Logo GeoTools. Převzato z [1]. 32
7.2 Struktura GeoTools. Převzato z [1]. 33
7.3 Logo JTS Topology Suite. Převzato z [7]. 33
7.4 Struktura projektu. (zdroj: autor) 34
A.1 Verze QGIS. (zdroj: autor) 42
A.2 Verze ArcGIS. (zdroj: autor) 43

Literatura

[1] GeoTools. https://www.geotools.org. cit. 2020-05-14.

[2] Oracle Java Documentation. https://docs.oracle.com/. cit. 2020-05-05.

[3] Open Source Geospatial Foundation. https://www.osgeo.org/. cit. 2020-05-01.

[4] ArcGIS Documentation. https://desktop.arcgis.com/en/documentation/. cit. 2020-05-21.

[5] Esri. https://www.esri.com/. cit. 2020-05-11.

[6] GNU. https://www.gnu.org/. cit. 2020-05-09.

[7] LocationTech. https://projects.eclipse.org/projects/locationtech. cit. 2020-05-03.

[8] Apache Maven Project. https://maven.apache.org/. cit. 2020-05-10.

[9] Záchrana pohřešovaných osob - pátrací akce v terénu. https://www.hzscr.cz/soubor/stc-zpat07-final-pdf.aspx. cit. 2020-05-01.

[10] BAYER, Tomáš. *Algoritmy v digitální kartografii.* : Karolinum, 2008.

[11] BENTLEY, Jon Louis a OTTMANN, Thomas A. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers.* 1979, , č. 9, s. 643–647.

[12] CHALOUPKOVÁ, Ivona SVOBODOVÁ a Vladimír MAKEŠ H. *Využití vyspělých technologií a čichových schopností psů pro zvýšení efektivity vyhledávání pohřešovaných osob v terénu (PÁTRAČ).* Praha: Ministerstvo vnitra České republiky, 2017.

[13] COUPLAND, Simon a JOHN, Robert. Geometric Type-1 and Type-2 Fuzzy Logic Systems. *Fuzzy Systems, IEEE Transactions on.* 03 2007, 15, s. 3 – 15. doi: 10.1109/TFUZZ.2006.889764.

[14] BERG, Mark et al. *Computational Geometry: Algorithms and Applications*. Berlin/Heidelberg: Springer Berlin Heidelberg, third edition, 2008. ISBN 9783642096815;3642096816;3540779736;9783540779735;.

[15] FRIGO, Matteo et al. Cache-oblivious algorithms. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 1999. s. 285–297.

[16] HARTMANIS, J. a STEARNS, R. E. On the Computational Complexity of Algorithms. *Transactions of the American Mathematical Society*. 1965, 117, s. 285–306. ISSN 00029947. Dostupné z: http://www.jstor.org/stable/1994208.

[17] JAN, Kolář. *Geografické informační systémy 10.* : Praha: ČVUT, 2003. ISBN 8001026876.

[18] JOAQUIM, Alfredo Ferreira Manuel J Fonseca a JORGE, A. Polygon detection from a set of lines. 2003.

[19] Katalog typových činností integrovaného záchranného systému. *Záchrana pohřešovaných osob - pátrací akce v terénu.* Česká republika, 2010. Dostupné z: https://www.hzscr.cz/soubor/stc-zpat07-final-pdf.aspx.

[20] MILKOVÁ, E et al. Algoritmy–základní konstrukce v příkladech a jejich vizualizace. *Hradec Králové, Gaudeamus*. 2010.

[21] PAVLIŠTA, Rostislav. *Řízení pátracích akcí po pohřešovaných osobách.* Ostrava: , 2009.

[22] PostGIS. *PostGIS - Spatial and Geographic objects for PostgreSQL*. Open Source Geospatial Foundation, 2019. Dostupné z: https://postgis.net/.

[23] QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2009. Dostupné z: http://qgis.osgeo.org.

[24] SLÁDKOVÁ, Vendula. *Aplikace pro tvorbu pátracích sektorů na základě přirozených bariér.* : , 2019.

[25] SOJKA, Eduard; NĚMEC, Martin a FABIÁN, Tomáš. Matematické základy počítačové grafiky, 2011.

[26] TUČEK, Ján. *Geografické informační systémy : principy a praxe.* : Computer Press, Vyd. 1. Praha, xiv., 364, 1998. ISBN 807226091X.

[27] KREVELD, Marc et al. *Algorithmic foundations of geographic information systems.* 1340. : Springer, 1997.

[28] VESELÝ, Martin. *TopoXML - výměnný formát topologie vektorových dat.* Disertační práce, Masarykova univerzita, Přírodovědecká fakulta, Brno, 2007. Dostupné z: `https://is.muni.cz/th/e65f2/`.

[29] WALFORD, Nigel. *Geographical data: characteristics and sources.* New York: John Wiley & Sons, 2002.

[30] WIRTH, Niklaus. *Algoritmy a štruktúry údajov.* Praha: Alfa, 1989.

[31] ZEMAN, Jiří. Návrh informačního portálu a informačního systému pro SAR (Search and Rescue) složky integrovaného záchranného systému, 2009.

A Srovnání polygonizace v ArcGIS a QGIS

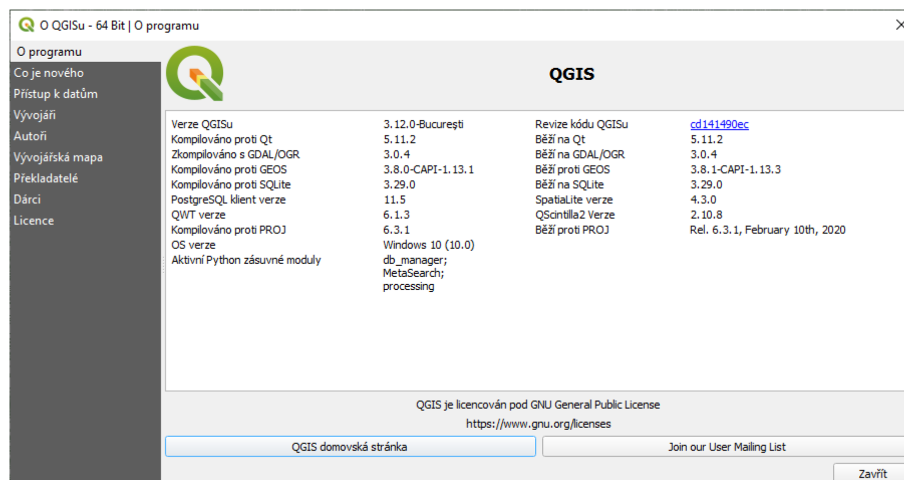
Srovnání bylo provedeno na testovacích datech, které poskytl *Ing. Jan Růžička, Ph.D.* Jelikož nemohla být poskytnuta reálná data, jedná se o uměle vygenerované linie, které jsou dle slov doktora Růžičky obdobného charakteru a velikosti, jako linie pro vytvoření pátracích sektorů v projektu Pátrač. Data obsahují celkem 10000 linií. Tato data byla dále testována ve zmenšené formě a to s 1000 a 100 liniemi.

A.1 Parametry počítače

- **Operační systém:** Windows 10 Enterprice LTSC
- **Výrobce:** HP
- **Model:** Z240
- **Procesor:** Intel(R) Core(TM) i7-7700 CPU @3.60GHz 3.60GHz, 64bit
- **RAM:** 32,0 GB

A.2 Výsledky a porovnání

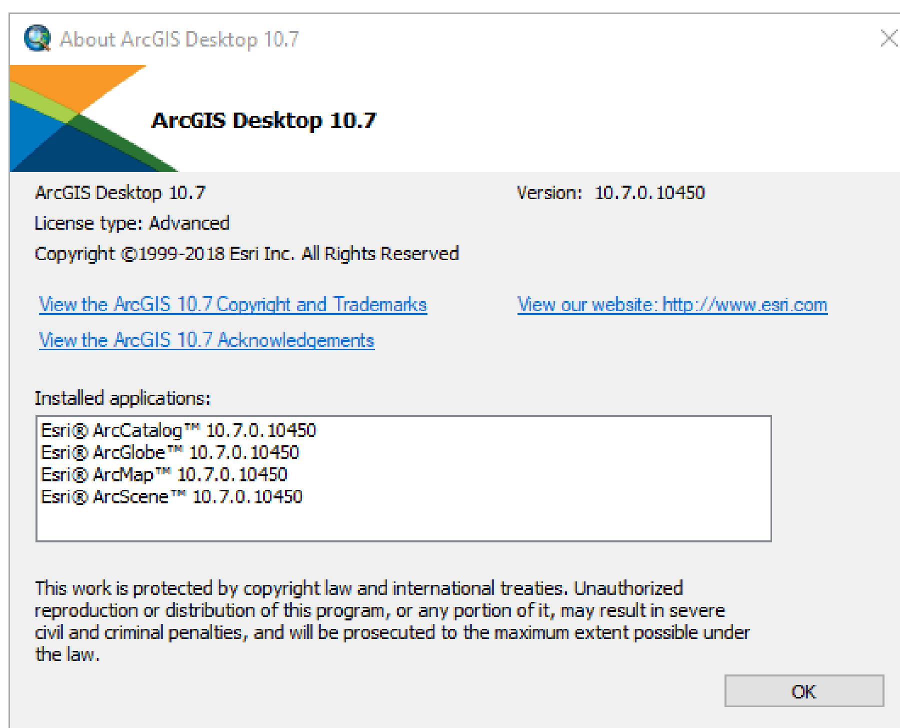
Pro každý dataset byla provedena 5x polygonizace v jednotlivých softwarech a výsledný čas byl spočítán jako aritmetický průměr. Na datech, která byla použita pro test, nám nevzniká nějaký zásadní rozdíl se zvětšujícím se datasetem a oba nástroje se chovají velmi podobně. Výsledky jsou uvedeny v tabulce A.1.



Obrázek A.1: Verze QGIS. (zdroj: autor)

ArcGIS						
linií	1. [s]	2. [s]	3. [s]	4. [s]	5. [s]	průměr
100	0,41	0,40	0,40	0,40	0,40	**0,40s**
1000	13,56	13,57	13,58	13,57	13,55	**13,57s**
10000	768	792	769	774	768	**12min 54s**
QGIS						
linií	1. [s]	2. [s]	3. [s]	4. [s]	5. [s]	průměr
100	0,16	0,06	0,07	0,06	0,06	**0,08s**
1000	10,02	9,95	9,98	9,96	9,97	**9,98s**
10000	748,00	768,21	751,74	753,92	757,29	**12min 36s**

Tab. A.1: Srovnání časů výpočtu polygonizace v ArcGIS a QGIS. (zdroj: autor)



Obrázek A.2: Verze ArcGIS. (zdroj: autor)

B Třída PolygonizeOp

```
package cz.ctu.gis.klemsa.polygonize;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import org.locationtech.jts.geom.Geometry;
import org.locationtech.jts.geom.GeometryCollection;
import org.locationtech.jts.geom.GeometryFactory;
import org.locationtech.jts.geom.LineString;
import org.locationtech.jts.geom.Polygon;
import org.locationtech.jts.geom.util.GeometryExtracter;
import org.locationtech.jts.operation.polygonize.Polygonizer;

/**
 * Polygonize a <code>Collection</code> of {@link Geometry}s or a
 ↪ single Geometry
 * (which may be a {@link GeoometryCollection}) together.
 * Only the {@link LineString}s will be included to polygonization.
 * {@link LineString}s may contain dangels. These will be added as
 ↪ part of the polygonization.
 *
 * @author Tomas Klemsa
 *
 */
public class PolygonizeOp
{

  private List lines = new ArrayList();

```

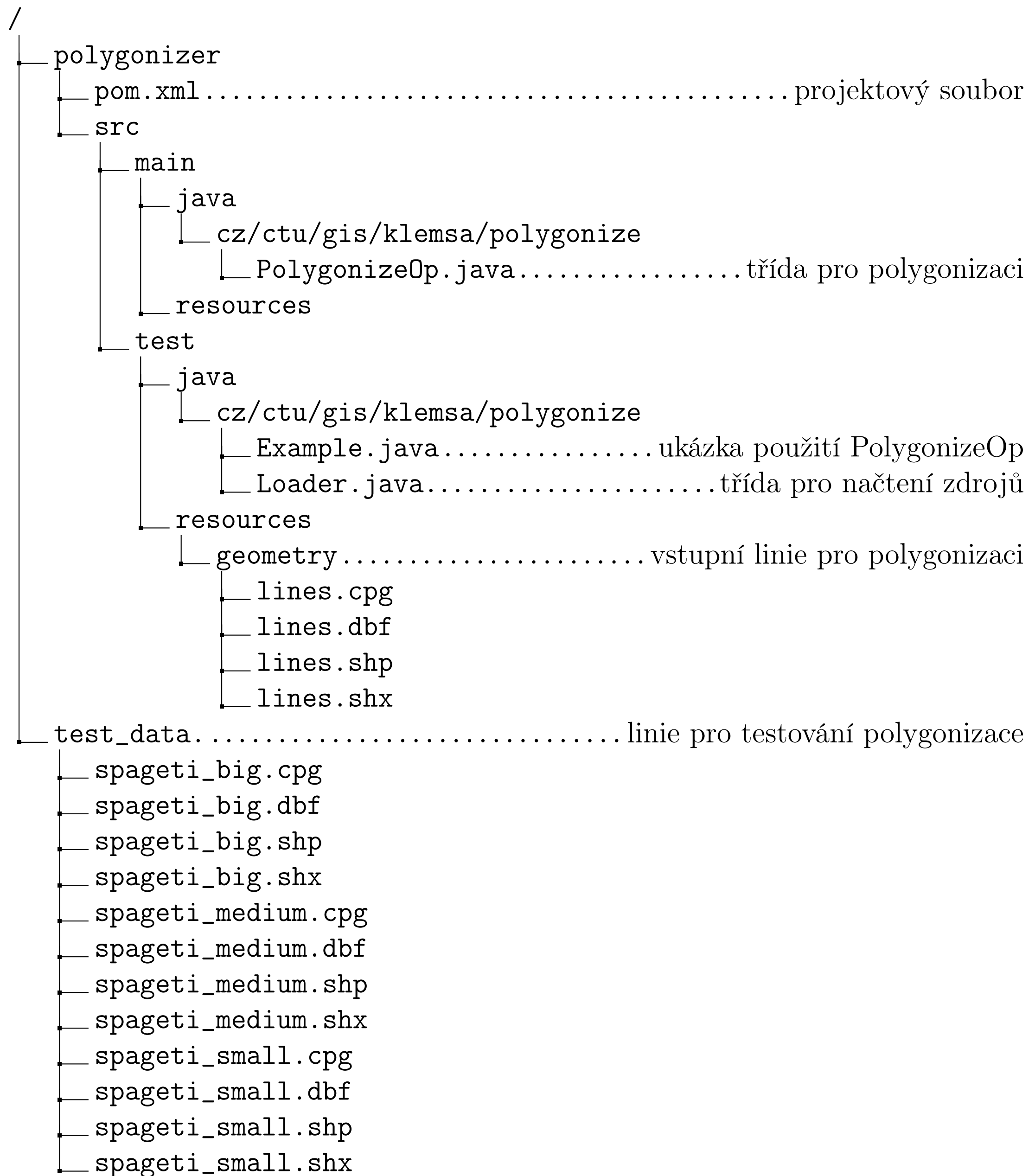
```
30    private GeometryFactory geomFact = null;
31
32    /**
33     * Constructs a polygons from a {@link Collection}
34     * of {@link Geometry}s.
35     *
36     * @param geoms a collection of geometries
37     * @param geomFact the geometry factory to use if the collection is
   ↪   empty
38     */
39    public PolygonizeOp(Collection geoms, GeometryFactory geomFact)
40    {
41      this.geomFact = geomFact;
42      extract(geoms);
43    }
44
45    /**
46     * Constructs a polygons from a {@link Collection}
47     * of {@link Geometry}s.
48     *
49     * @param geoms a collection of geometries
50     * @return the union of the geometries,
51     * or <code>null</code> if the input is empty
52     */
53    public static Collection polygonize(Collection geoms)
54    {
55      PolygonizeOp op = new PolygonizeOp(geoms);
56      return op.polygonize();
57    }
58
59
60    /**
61     * Constructs a polygons from a {@link Collection}
```

```
62      * of {@link Geometry}s.
63      *
64      * If no input geometries were provided but a {@link
    ↪   GeometryFactory} was provided,
65      * an empty {@link GeometryCollection} is returned.
66        *
67      * @param geoms a collection of geometries
68      * @param geomFact the geometry factory to use if the collection is
    ↪   empty
69      * @return the union of the geometries,
70      * or an empty GEOMETRYCOLLECTION
71      */
72     public static Collection polygonize(Collection geoms,
        ↪   GeometryFactory geomFact)
73     {
74       PolygonizeOp op = new PolygonizeOp(geoms, geomFact);
75       return op.polygonize();
76     }
77
78     /**
79      * Constructs a polygons from a {@link Geometry}
80      * (which may be a {@link GeometryCollection}).
81      *
82      * @param geom a geometry to union
83      * @return the union of the elements of the geometry
84      * or an empty GEOMETRYCOLLECTION
85      */
86     public static Collection polygonize(Geometry geom)
87     {
88       PolygonizeOp op = new PolygonizeOp(geom);
89       return op.polygonize();
90     }
91
```

```
92    /**
93     * Constructs a polygons from a {@link Collection}
94     * of {@link Geometry}s, using the {@link GeometryFactory}
95     * of the input geometries.
96     *
97     * @param geoms a collection of geometries
98     */
99    public PolygonizeOp(Collection geoms)
100   {
101     extract(geoms);
102   }
103
104   /**
105    * Constructs a polygons from a {@link Geometry}
106    * (which may be a {@link GeometryCollection}).
107    * @param geom
108    */
109   public PolygonizeOp(Geometry geom)
110   {
111     extract(geom);
112   }
113
114   private void extract(Collection geoms)
115   {
116     for (Iterator i = geoms.iterator(); i.hasNext();) {
117       Geometry geom = (Geometry) i.next();
118       extract(geom);
119     }
120   }
121
122   private void extract(Geometry geom)
123   {
124     if (geomFact == null)
```

```
125         geomFact = geom.getFactory();
126
127       // Extract only lineStrings.
128       GeometryExtracter.extract(geom, LineString.class, lines);
129     }
130
131     /**
132      * Gets the list of polygons formed by the polygonization.
133      * @return a collection of {@link Polygon}s
134      */
135     public Collection polygonize()
136     {
137       Polygonizer polygonizer = new Polygonizer();
138       Geometry lineGeom = geomFact.buildGeometry(lines);
139
140       // Calculate nodes.
141       Geometry unionLines = lineGeom.union();
142       System.out.println(unionLines);
143
144       // Calculate polygons.
145       polygonizer.add(unionLines);
146
147       return polygonizer.getPolygons();
148     }
149   }
```

C Elektornické přílohy



```
/
├── polygonizer
│   ├── pom.xml ........................................ projektový soubor
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── cz/ctu/gis/klemsa/polygonize
│       │   │       └── PolygonizeOp.java ................ třída pro polygonizaci
│       │   └── resources
│       └── test
│           ├── java
│           │   └── cz/ctu/gis/klemsa/polygonize
│           │       ├── Example.java ..................... ukázka použití PolygonizeOp
│           │       └── Loader.java ...................... třída pro načtení zdrojů
│           └── resources
│               └── geometry ............................. vstupní linie pro polygonizaci
│                   ├── lines.cpg
│                   ├── lines.dbf
│                   ├── lines.shp
│                   └── lines.shx
└── test_data ............................................ linie pro testování polygonizace
    ├── spageti_big.cpg
    ├── spageti_big.dbf
    ├── spageti_big.shp
    ├── spageti_big.shx
    ├── spageti_medium.cpg
    ├── spageti_medium.dbf
    ├── spageti_medium.shp
    ├── spageti_medium.shx
    ├── spageti_small.cpg
    ├── spageti_small.dbf
    ├── spageti_small.shp
    └── spageti_small.shx
```