Gymnázium, Praha 6, Arabská 14
Programování

Maturitní práce

2025

Michal Bogdanov

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

Maturitní práce

Předmět: Programování
Téma: Analyzátor Prahy

Školní rok: 2024/2025

Autor: Michal Bogdanov
Třída: 4. E
Vedoucí práce: Mgr. Jan Lána
Třídní učitel: Mgr. Blanka Hniličková

Na tomto místě bych chtěl poděkovat Mgr. Janu Lánovi za konzultace a vedení projektu.

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne …… 2025 podpis: ……

Michal Bogdanov

Anotace

Tato práce představuje aplikaci k analýze služeb Prahy pomocí Voroného diagramu. Řeší problémy spojené s Voroného diagramem v programování jako je Fortunův algoritmus. Stejně tak polemizuje o převodu souřadnic do souřadnic aplikace na základě vzorců.

Annotation

Project presents an application to analyze the services of Prague, using a Voronoi diagram. It solves the problems associated with Voronoi diagram in programming such as Fortune's algorithm. As well as it discusses the conversion of coordinates to application coordinates based on formulas.

Zadání projektu:

Analyzátor Prahy

Cílem projektu je vytvořit aplikaci, která umožní uživatelům vizuální analýzu Prahy. Aplikace bude analyzovat vybrané služby v Praze jako je např. nemocnice, školy, apod. pomocí implementace Voroného diagramu. Mapa Prahy bude rozdělena do oblastí na základě služeb.

Voroného diagramy vytváří oblasti okolo bodů. V takové oblasti se pak nachází množina bodů, které jsou nejblíže k danému bodu. Aplikace bude mapa, kde bude možnost vybrat několik veřejných služeb. Program získá souřadnice služby a mapu následně rozdělí do oblastí - neznamená to nejkratší cesta k bodu.

Obsah

1 Úvod

Cílem této práce je vytvořit software, který využívá Voroného diagram k analýze dostupnosti veřejných služeb na území Prahy. Voroného diagram je matematický model rozdělení prostoru na oblasti podle vzdálenosti k vybraným bodům. V tomto případě budou těmito body různé veřejné služby, jako jsou lékárny, stanice metra a autobusové zastávky. Každá oblast bude přiřazena k nejbližší službě, čímž vznikne přehledná vizualizace jejich rozmístění a dostupnosti.

Hlavním přínosem tohoto projektu je možnost podrobné analýzy dostupnosti služeb v různých částech města. Díky vytvořenému systému bude možné určit, které oblasti mají dostatečné pokrytí například zdravotnickými zařízeními nebo veřejnou dopravou, a naopak identifikovat místa, kde je přístup ke službám omezený. Tato analýza může být užitečná nejen pro obyvatele města, kteří hledají nejbližší služby, ale i pro městské plánovače, kteří chtějí optimalizovat infrastrukturu a zajistit rovnoměrnější rozložení veřejných služeb.

Projekt se zaměřuje na kombinaci Voroného diagramu s reálnými geografickými daty, přičemž vizualizace výsledků bude zobrazena nad mapou Prahy. Tento přístup umožní snadnou interpretaci výsledků a pochopení rozložení jednotlivých oblastí. Mapová reprezentace pomůže uživatelům lépe porozumět dostupnosti služeb a identifikovat případné problematické oblasti, kde by mohlo být vhodné zlepšit zajištění infrastruktury. Důležitou součástí práce je také způsob získávání a zpracování dat o veřejných službách, přičemž projekt využije dostupné databáze a API, jako je Overpass API pro získávání bodů zájmu z OpenStreetMap.

Kromě vizuální stránky se projekt ještě zaobírá stránkou teoretickou v rámci algoritmů a matematiky.

2 VORONÉHO DIAGRAMY

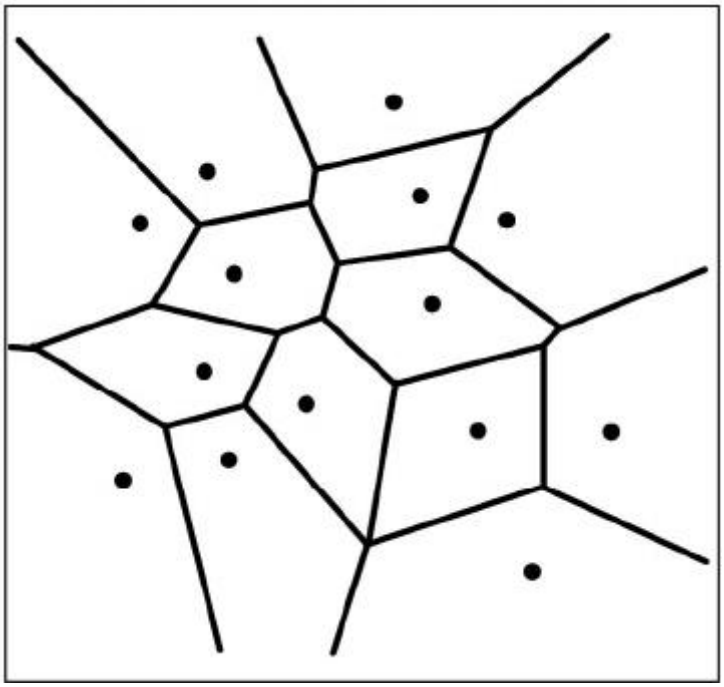
Pojmenované po Georgijem Feodosjeviči Voroném, který byl matematik působící koncem 19. století a začátkem 20. století v Rusku, na Ukrajině a v Polsku.[1] Diagram znázorňuje rozdělení roviny na oblasti, o kterých platí, že oblast okolo bodu tvoří množinu bodů, které jsou blíže k tomuto bodu než je jakýkoli další bod z jiné oblasti. [6]

Matematický zápis definice zní takto:

Nechť $S = \{p_1, p_2, \ldots, p_n\}$ je množina bodů v metrickém prostoru s danou vzdálenostní funkcí $d(x,y)$. Voroného oblast bodu p_i je množina bodů, které jsou blíže k p_i než ke kterémukoli jinému bodu $p_j (j \neq i)$:

$$V(p_i) = \{x \in \mathbf{R}^2 \mid d(x,p_i) < d(x,p_j) \text{ pro všechna } j \neq i\}$$

Voroného diagram je pak rozdělení prostoru na Voroného oblasti odpovídající jednotlivým bodů v S.[2] Vytvořené diagramy se uplatňují v několika oborech jako je geometrie, přírodní vědy, sociologie, hydrologie nebo dokonce v modelování struktury v 3D tisku apod..[6] Ve vědách se takové rozdělení používá k analýze či plánování systému.



Obrázek 1 Voroného diagram [17]

2.1 Historie

Koncept Voroného diagramů sahá hluboko do historie a jejich základní principy lze vystopovat až do 17. století. Ačkoli samotný pojem „Voroného diagram“ vznikl až později, jeho podstata

byla v různých formách využívána již mnohem dříve. René Descartes, slavný francouzský filozof a matematik, ve svém díle „Principy filozofie“ z roku 1644 neformálně použil myšlenku rozdělení prostoru, která byla v mnohém podobná Voroného diagramům. Descartes použil tento koncept k ilustraci rozložení hmoty ve sluneční soustavě a jejím okolí, přičemž předpokládal, že prostor může být rozdělen na oblasti kolem jednotlivých nebeských těles v závislosti na jejich přitažlivosti a vzájemném působení. Ačkoli tehdy nebyly Voroného diagramy definovány v dnešním matematickém smyslu, jednalo se o jeden z prvních historických příkladů jejich využití.[5]

Další významná aplikace podobné myšlenky se objevila v 19. století, kdy britský fyzik John Snow použil grafickou metodu, která se strukturou velmi podobala Voroného diagramům, k analýze šíření cholery v Londýně. Během velké cholery v roce 1854 si Snow povšiml, že výskyt onemocnění nebyl v městských čtvrtích rovnoměrně rozprostřen, ale naopak se soustředil do určitých oblastí. Rozhodl se tedy zakreslit polohu jednotlivých případů cholery na mapě Londýna a porovnat ji s umístěním studen, které tehdy sloužily jako hlavní zdroj pitné vody pro obyvatele. Výsledná analýza ukázala, že nejvíce nakažených osob žilo v blízkosti jedné konkrétní studny, která se nakonec ukázala být zdrojem nákazy. Tímto způsobem Snow dokázal spojit šíření cholery se znečištěnou vodou. Jeho myšlenka byla blízká k myšlence rozdělení prostoru na oblasti, kde každý bod patří k nejbližšímu zdroji, a tak Voroného diagramy fungují.[5]

Struktury, které dnes nesou označení Voroného diagramy, získaly své jméno po významném ruském matematikovi Georgiji Feodosjeviči Voroném. Voronoj se ve své práci intenzivně věnoval studiu geometrických vlastností prostoru a jejich matematickému popisu. Jedním z jeho nejvýznamnějších přínosů bylo to, že se mu podařilo zobecnit a přesně nadefinovat Voroného diagramy pro n-rozměrné prostory, což znamená, že rozšířil jejich původní dvourozměrný koncept do libovolného počtu dimenzí. Tím vytvořil pevný matematický základ, na kterém později stavěly další generace vědců a matematiků. Díky jeho práci bylo možné Voroného diagramy nejen lépe pochopit a analyzovat, ale také aplikovat v různých oblastech, kde jejich využití přináší významné výsledky. [1] [5]

2.2 Přírůstkový algoritmus

Přírůstkový algoritmus je jednou z nejjednodušších metod pro konstrukci Voroného diagramu a je někdy označován jako „Naivní algoritmus“, protože jeho základní implementace je poměrně triviální, avšak ve své základní podobě postrádá efektivitu. Přestože nepatří mezi nejrychlejší metody, jeho výhoda spočívá v jednoduchosti implementace a snadném pochopení jeho principu.[5] Tento algoritmus pracuje tak, že nejprve vytvoří malou podmnožinu Voroného diagramu, obvykle obsahující pouze několik bodů, a poté postupně přidává další body, přičemž se s každým novým bodem upravuje již existující diagram. Tímto způsobem se postupně vytváří celý Voroného diagram.

Každým přidáním nového bodu do existujícího diagramu dochází k jeho aktualizaci, což znamená, že je třeba určit oblasti, které budou změnou ovlivněny. Konkrétně je nutné zjistit, které části stávajícího Voroného diagramu se změní, protože právě přidaný bod bude mít v

některých oblastech nejkratší vzdálenost ve srovnání s ostatními body. Následně je nutné tyto změněné oblasti přepočítat a aktualizovat hranice tak, aby každý bod v dané oblasti patřil k nejbližšímu bodu diagramu. Tento proces se opakuje s každým dalším přidaným bodem, což znamená, že postupně dochází k rozšiřování a upravování Voroného diagramu.[3]

Jedním z největších problémů tohoto přístupu je jeho výpočetní neefektivita, zejména pro velké množství bodů. Při každém přidání nového bodu je nutné prohledat celý existující Voroného diagram, což vede k velmi vysoké výpočetní náročnosti. Časová složitost tohoto algoritmu je $O(n^2)$, což znamená, že s rostoucím počtem bodů roste doba výpočtu kvadraticky. Pro menší množiny dat nemusí být tento problém příliš patrný, avšak pokud pracujeme s velkými datovými sadami, jako jsou například souřadnice stovek nebo tisíců městských služeb v geografickém informačním systému, může být tento přístup velmi pomalý a nepraktický.[3]

Za další nevýhodu přírůstkového algoritmu lze považovat skutečnost, že je častá potřeba opravovat strukturu diagramu při přidávání nových bodů. Každý nový bod totiž může ovlivnit poměrně rozsáhlou oblast, což vede k nutnosti opětovného výpočtu několika hran a průsečíků. Vzhledem k tomu, že přepočítání diagramu probíhá po každém přidání nového bodu, dochází k neefektivnímu využití výpočetních zdrojů, což ještě více umocňuje jeho výpočetní náročnost.[4]

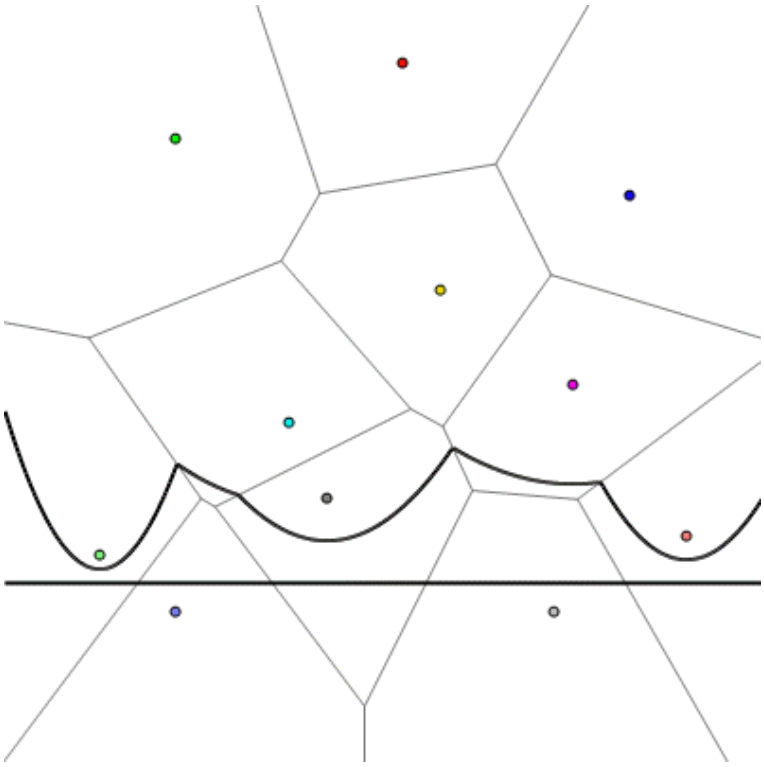
Přestože je tento algoritmus poměrně jednoduchý na implementaci, jeho využití je vhodné pouze pro menší množiny bodů, kde rychlost výpočtu není kritickým faktorem. V případě větších datasetů je vhodné hledat efektivnější možnosti například Fortunův algoritmus.

2.3 Fortunův algoritmus

Fortunův algoritmus představuje poměrně složitější, ale zato velmi efektivní metodu konstrukce Voroného diagramu, která využívá koncept „zametání“ roviny. Tuto metodu navrhl Steven Fortune v roce 1986 a dodnes patří mezi nejefektivnější způsoby generování Voroného diagramu.[7] Název „zametací algoritmus“ vychází z principu jeho fungování, kdy se v rovině pohybuje horizontální přímka shora dolů (v našem případě naopak) a při každém průchodu bodem vytváří tzv. událost. Přímka postupně rozděluje rovinu na zpracovanou a nezpracovanou část a je často označována jako „pobřežní čára“ nebo „plážová čára“ (beachline). Tato čára reprezentuje hranici mezi oblastmi, které již byly zpracovány, a těmi, které na zpracování teprve čekají.[8]

Fortunův algoritmus pracuje se dvěma typy událostí a to jsou místní události (site events) a kružnicové události (circle events). Místní událost nastává v okamžiku, kdy zametací přímka dosáhne nového bodu, což vede ke vzniku nové paraboly v diagramu. Tato parabola má svůj vrchol v daném bodě a v závislosti na poloze zametací přímky se postupně rozšiřuje. Zametací přímka zároveň funguje jako řídicí přímka, která definuje zakřivení paraboly. Druhým typem události je kružnicová událost, která nastává v okamžiku, kdy některá parabola zaniká, což vede

ke vzniku nové hrany ve Voroného diagramu. Když taková událost nastane, odstraní se odpovídající parabola a vzniklý průsečík se zaznamená jako nový Voroného vrchol.[7][9]



Obrázek 2 Fortunův algoritmus s plážovou čárou [18]

Postup algoritmu:

1. Nejprve se všechny místní události (tj. souřadnice bodů, které reprezentují služby, jako jsou lékárny, stanice metra či autobusové zastávky) vloží do prioritní fronty. Tato fronta je seřazena podle y-souřadnice v sestupném pořadí, což odpovídá pohybu zametací přímky zdola nahoru, protože JavaFX používá souřadnice obráceně.

2. Algoritmus zpracuje postupně události, které vybírá z fronty. Pokud je událost místní, vytvoří se nová parabola, která se přidá do plážové čáry. Pokud přidání nové paraboly způsobí vytvoření kružnicové události (např. když se v daném místě vytvoří tři sousedící paraboly), pak se tato kružnicová událost přidá do fronty ke zpracování. Když nastane kružnicová událost, znamená to, že některá parabola zanikla. V tomto okamžiku algoritmus odstraní příslušnou parabolu z plážové čáry a přidá nový průsečík (vrchol) do Voroného diagramu. Dále se propojí vzniklé hrany, čímž se postupně konstruuje celý diagram.

Fortunův algoritmus tvoří časovou složitost ***O(nlog(n))***, což umožnuje tvořit Voroného diagram i pro větší objem dat. [7][9]

2.4 Implementace

Pro svojí aplikaci jsem zvolil Fortunův algoritmus pro konstrukci Voroného diagramu. Hlavním důvodem této volby byla jeho efektivita. Fortunův algoritmus je schopen vypočítat Voroného diagram v čase $O(n \log n)$, což je důležité zejména při práci s velkým množstvím bodů (služeb) na mapě. Při použití méně efektivních algoritmů by výpočet Voroného diagramu trval mnohem déle, což by negativně ovlivnilo uživatelskou zkušenost a použitelnost aplikace.

Přestože je Fortunův algoritmus dobře známý a široce využívaný, v programovacím jazyce Java existuje velmi málo funkčních nebo použitelných implementací tohoto algoritmu. Většina dostupných zdrojů buď neposkytovala plně funkční kód, nebo byly implementace příliš zjednodušené a občas nesrozumitelné, nebo se nehodily k implementaci do mé aplikace. Podařilo se mi však najít funkční Fortunův algoritmus, ale pouze pro základní případy.[10] Ovšem některé speciální případy vyřešené nebyly. Rozprostření bodů, které přinášejí problémy, je podrobněji zobrazeno na této stránce https://www.algovision.org/Algovision/pool.html, přičemž řešení jednoho problému, vyřešilo i některé další.

Při implementaci Fortunova algoritmu jsem narazil na **problém se svislou čarou**, který vzniká ve chvíli, kdy více bodů (služeb) sdílí stejnou y-souřadnici. Jelikož plážová čára jde v kartézské soustavě od shora dolu, ale v JavaFX zdola nahoru tedy $y_1 < y_2$, nastává problém při výpočtu hran a jejich průsečíků. Pro výpočet využívám předpisu pro lineární funkci, jinak řečeno směrnicový předpis $y = kx + q$, kde $k = tg(\alpha)$ a α je úhel, který svírá přímka s osou x. Tangens však není definovaný při 90º neboli, když přímka je $x = konstantě$. To znamená, že pokud x zůstává konstantní, nemohu použít běžný předpis lineární funkce, protože směrnicová rovnice se pro svislé čáry nedá použít.

Možné řešení:

1. Průsečík dvou přímek lze spočítat pomocí dvou rovnic o dvou neznámých. Pokud směrnice k není definovaná, což nastává u svislých přímek, pak mají rovnice tvar $x = konst.$ a $y = kx + q$. V případě, že jsou přímky rovnoběžné nebo dokonce identické, nemá smysl jejich průsečík vůbec počítat, protože buď žádný neexistuje, nebo mají nekonečně mnoho společných bodů. Standardně lze tedy určit x-souřadnici a následně dopočítat y-souřadnici průsečíku, protože podmínka $x = konst.$ jasně definuje hodnotu x, která se nemění. Toto řešení se na první pohled jeví jako správné, ale v praxi se objevují problémy zejména při vykreslování výsledného Voroného diagramu. Začnou se generovat „nesmyslné“ čáry, které nereprezentují správné rozdělení prostoru, a tím narušují vizuální i logickou interpretaci výstupu algoritmu.

2. Existuje i o něco jednodušší řešení tohoto problému. Stačí vytvořit funkci, která mírně posune body se stejnou y-souřadnicí o velmi malou hodnotu epsilon, což zajistí, že přímka rozdělující body nebude zcela svislá, ale bude mít alespoň minimální sklon. Tento sklon je natolik zanedbatelný vzhledem k celkové velikosti mapy, že nemá žádný viditelný dopad na výslednou vizualizaci. Hodnota epsilon je extrémně malá, obvykle v řádu 10^{-10}, což znamená, že rozdíl mezi původními a upravenými souřadnicemi je pro

lidské oko nepostřehnutelný a nemění správnou interpretaci dat. Tento přístup nejenže eliminuje problém s nedefinovaným tangens u svislých přímek, ale zároveň zachovává přesnost výpočtů bez nežádoucích chyb. Navíc při implementaci této metody je důležité odstranit duplicitní body, protože jejich přítomnost by mohla vést k podobným problémům, jaké se vyskytovaly v předchozím řešení. Duplicitní body by mohly způsobovat nesprávné propojení hran, což by vedlo k chybnému vykreslení Voroného diagramu a vzniku nesmyslných spojnic. Tímto jednoduchým trikem se tedy lze vyhnout složitějším opravám geometrických nesrovnalostí a zajistit plynulé fungování algoritmu bez viditelných defektů v konečné vizualizaci.

```
private void perturbPoints() {
    double epsilon = 1e-10;
    Map<Double, Integer> yCount = new HashMap<>();

    for (Point p : sites) {
        if (yCount.containsKey(p.y)) {
            yCount.put(p.y, yCount.get(p.y) + 1);
            p.y += yCount.get(p.y) * epsilon;
        } else {
            yCount.put(p.y, 0);
        }
    }
}
```

Ukázka 1 Posouvání bodu o epsilon

Další problém nastal u **nekonečně dlouhých hran**, které vznikaly v důsledku vlastností Voroného diagramu. Některé hrany nemají průsečíky v rámci zobrazené mapy, což znamenaná, že by pokračovaly nekonečně daleko a byly ukončeny až hranicemi okna programu. To by však vedlo k velkým uživatelským nepříjemnostem, protože uživatel by musel zdlouhavě posouvat bílou plochou po celé aplikaci, aby vůbec našel hrany nebo mapu. Na druhou stranu se na mapě nacházely hrany, které nevedly až na okraj mapy. Dále se v diagramu objevovaly i hrany, které sice byly ohraničené, ale nepokrývaly část zobrazené mapy, což by z pohledu uživatele, bylo zbytečně matoucí.

Řešení:

Pro správnou úpravu vykreslení Voroného diagramu bylo zapotřebí několik kroků. Prvním klíčovým bodem bylo přenastavit proměnnou *currentY* v metodě *generateVoronoi()* tak, aby její hodnota odpovídala součtu výšky a šířky mapy. Tento krok zajistil, že všechny hrany, které měly zasahovat až k okrajům mapy, skutečně dosáhly svého konce a nebyly zbytečně ukončeny dříve.

Další důležitou součástí řešení bylo nalezení průsečíku nekonečných hran s okraji mapy. Pro výpočet těchto bodů byl využit podobný přístup jako u prvního řešení předchozího problému se svislou čárou, ale tentokrát s tím rozdílem, že konstantní přímky tvořily samotné hranice mapy. Pro nalezení průsečíku mezi nekonečnou hranou a okrajem mapy je funkce *setEnd()*, která se v případě potřeby rekurzivně volá znovu, aby se ověřilo, zda nevznikl ještě jiný průsečík, který by měl být zohledněn. Pokud byl nalezen průsečík odpovídající velikosti mapy, stal se tento bod koncovým bodem dané hrany, což zajistilo správné ukončení nekonečných přímek.

Nicméně problém se netýkal pouze koncového bodu některých hran, ale i jejich počátečních bodů. Některé přímky totiž nezačínaly v rámci viditelné části mapy, ale sahaly až do nekonečna, což vedlo k tomu, že jejich počátek nebyl vůbec definován a aplikace vytvořila vlastní počátek s vysokými nebo naopak velmi nízkými (zápornými) hodnotami na souřadnicích. Aby se tento problém eliminoval, byla implementována kontrola do konstruktoru, která zjistila, zda počátek hrany přesahuje hranici mapy. Pokud tomu tak bylo, upravila se souřadnice tak, aby hrana začínala přesně na okraji mapy. Pokud počáteční bod přesahoval x-ovou hranici mapy, nastavila se jeho x-souřadnice buď na nulu (pro levý okraj), nebo na maximální šířku mapy (pro pravý okraj). Podobně, pokud počáteční bod přesahoval y-novou hranici, nastavila se y-souřadnice na nulu (pokud hrana začínala mimo horní hranici mapy) nebo na maximální výšku mapy (pokud přesahovala dolní okraj mapy). Tento postup zajistil, že všechny hrany měly jasně definovaný počáteční i koncový bod a žádná přímka již nepřesahovala vykreslenou oblast, ani se neztrácela v nekonečnu.

Další optimalizací bylo správné zpracování místních událostí ve Voroného algoritmu. Ačkoli se k jejich správnému uspořádání využívá datová struktura *PriorityQueue*, bylo vhodné provést ještě předběžné seřazení těchto událostí podle y-souřadnice. Díky tomu se zajistilo, že se události ve frontě objevovaly v logickém pořadí a nebyly přeházené, což minimalizovalo výpočetní komplikace při jejich zpracování a vedlo k plynulejšímu běhu celého algoritmu.

Celkové řešení tohoto problému nejenže zvýšilo přesnost a spolehlivost Voroného diagramu, ale také výrazně zlepšilo uživatelskou přívětivost aplikace. Díky správnému ukončení nekonečných hran a optimalizovanému uspořádání událostí se zajistilo, že mapa byla vizuálně přehledná a všechny oblasti byly správně vymezené, což uživatelům umožnilo snadněji analyzovat dostupnost služeb v různých částech Prahy.

3 OPENSTREETMAP

OpenStreetMap je open source projekt, tedy že zdrojový kód je otevřený a volně přístupný. Projekt je zaměřený na tvorbu a poskytování geografických dat na mapových podkladech včetně zeměpisných souřadnic. Projekt byl založen v červenci 2004 Stevem Coastem ve Velké Británii a od té doby se rozrostl v celosvětovou komunitu dobrovolníků, kteří společně vytvářejí a udržují aktuální mapová data. [11]

Hlavním cílem OSM je překonat omezení tradičních mapových služeb, které často podléhají přísným licenčním podmínkám a nejsou snadno dostupné pro veřejnost. Díky otevřené licenci Open Database License mohou uživatelé tato data volně používat, upravovat a sdílet, což podporuje inovace a široké spektrum aplikací v různých oblastech, od vývoje mobilních aplikací po vědecký výzkum.[11]

Komunita OSM je tvořena jednotlivci i organizacemi, kteří přispívají svými znalostmi a zkušenostmi k neustálému zlepšování mapových podkladů. Tito přispěvatelé využívají různé zdroje dat, včetně GPS zařízení, leteckých snímků a terénních pozorování, aby zajistili co nejpřesnější a nejaktuálnější informace. Díky této kolektivní snaze se OSM stala důvěryhodným zdrojem geografických informací pro mnoho uživatelů po celém světě. [12]

V České republice existuje aktivní komunita OpenStreetMap, která se podílí na mapování a údržbě dat v rámci projektu. Tato komunita nabízí různé mapové vrstvy a tematické mapy, které jsou dostupné na jejich webových stránkách. Uživatelé zde mohou také najít návody, jak se zapojit do projektu, a informace o dostupných aplikacích využívajících data OSM, jako je například aplikace Organic Maps pro offline zobrazení map a vyhledávání tras. [13]

3.1 Overpass API

Je důležité ještě zmínit nástroj jménem Overpass API, který umožňuje získávat a filtrovat data z OpenStreetMap (OSM) podle různých kritérií. Tento API umožňuje uživatelům dotazovat se na konkrétní objekty, jako jsou budovy, silnice, parky nebo specifické služby (například restaurace či bankomaty), a získávat pouze relevantní informace bez nutnosti stahování celé databáze OSM. Díky tomu je Overpass API efektivním řešením pro vývoj aplikací, které potřebují pracovat s geografickými daty.

Výhodou tohoto API je jeho dotazovací jazyk, který dokáže přijímat dotazy s využitím filtrů, podmínek a geografických omezení. Uživatelé mohou například snadno získat seznam všech kaváren v určitém městě, hledat cyklostezky nebo analyzovat dopravní infrastrukturu. Strukturu dotazů lze připravit přes webové rozhraní Overpass Turbo, které poskytuje vizuální zpětnou vazbu.[14]

3.2 Implementace

OpenStreetMap implementuji dvěma způsoby. První z nich je, že **pomocí webových adre**s typu *https://tile.openstreetmap.org/PŘIBLÍŽENÍ/X-SOUŘADNICE/Y-SOUŘADNICE.png* získám jednotlivé části mapy jako obrázky. Tento přístup umožňuje načítání mapových podkladů přímo ze serverů OpenStreetMap (OSM) v podobě dlaždic, které pokrývají určité geografické oblasti.

Abych dostal část mapy, kterou potřebuji, přepočítám hraniční souřadnice Prahy na souřadnice OSM. Souřadnice pro OpenStreetMap závisí na přiblížení mapy. Pro převod jsem napsal funkci *getIntTile(),* která jako vstupní parametry přijímá úroveň přiblížení a geografické souřadnice bodu ve světovém souřadnicovém systému (zeměpisná šířka a délka). Funkce následně provede transformaci těchto souřadnic na odpovídající x a y souřadnice dlaždic v OSM.

Úroveň přiblížení jsem nastavil jako konstantní s hodnotou 13, protože při tomto měřítku je mapa dostatečně detailní a zároveň přehledná. Použití pevného přiblížení má také praktické důvody např. dynamická změna zoomu by vedla k velmi náročným výpočtům a zbytečně dlouhým časům načítání mapových dlaždic ze serveru. Zakotvený přiblížení tedy zajišťuje optimální poměr mezi kvalitou zobrazení a výkonem aplikace.

Vzorec pro výpočet převodu[15]:

$$x = \left\lfloor \frac{lon + 180}{360} \cdot 2^{přiblížení} \right\rfloor$$

$$y = \left\lfloor (1 - \frac{\ln(\tan(lat \cdot \frac{\pi}{180}))}{\pi}) \cdot 2^{přiblížení-1} \right\rfloor$$

Rovnice 1 Převod na dlaždice

Při výpočtu souřadnic dlaždic v OpenStreetMap využívám specifický vzorec, který převádí zeměpisnou délku (lon) a šířku (lat) ze světového souřadnicového systému na souřadnice dlaždicového systému OSM. Jelikož se ve většině matematických funkcí pracuje s úhly v radiánech, je nutné před samotným výpočtem převést zeměpisné souřadnice ze stupňů na radiány.

Vzorec využiji pro dva klíčové body, které určují hranice oblasti a to severovýchodní roh a jihozápadní roh hranice Prahy. Body definují oblast dlaždic, kterou využiji pro zobrazení. Následně se pro zobrazení dotazuji v těchto intervalech tvořící obdélníkovou plochu $\langle x - jihozápad, x - severovýchod\rangle$ *a* $\langle y - jihozápad, y - severovýchod\rangle$.

Volba právě jihozápadního a severovýchodního rohu není náhodná. Důvodem je skutečnost, že souřadnice na jih a západ mají menší hodnoty než jejich protějšky na severu a východě. Díky této vlastnosti se dají souřadnicové intervaly jednodušeji spravovat a implementace kódu pro stahování dlaždic je přehlednější a efektivnější.

Po výpočtu souřadnic se jednotlivé dlaždice přidávají do „tabulky“ dlaždic, kde jsou uspořádány ve správném pořadí podle jejich polohy v mapě. Každá dlaždice v systému OpenStreetMap má pevně danou velikost, která je vyjádřena v pixelech 256×256. Aby bylo možné správně vykreslit celou mapu v okně aplikace, je nutné vypočítat, kolik dlaždic bude potřeba na výšku a na šířku.

Počet není důležitý pouze pro správnou velikost okna aplikace, ale také je potřebný pro další matematické operace. Příkladem jsou nekonečné hrany u Fortunova algoritmu, kde je důležité znát hranici mapy a ukončit tak hranu správně.

Jako **druhý** způsob implementuji **Overpass API**, které využívám pro získání souřadnic služeb. Pro základ této aplikace vyhledávám souřadnice stanic metra, lékáren a autobusových zastávek. S tímto způsobem získávání dat přichází několik problémů.

3.2.1 Stejnojmenné stanice

Pro stanice metra a autobusové zastávky vytvářím *HashMap*, což je datová struktura umožňující ukládání klíčů a hodnot, přičemž každý klíč je unikátní, tím eliminuji stejnojmenné názvy zastávek.

V případě autobusových zastávek se často stává, že jedna zastávka má dvě různé polohy pro stejný název. Jejich odlišnost pouze spočívá ve směru jízdy. Zastávky jsou přitom umístěny kousek od sebe a vytváření Voroného diagramu pro každou z nich by bylo zbytečné, když vyjadřují v podstatě stejnou věc. Vybere se tedy pouze jedna z nich.

Podobný problém nastal i u stanice metra, kde v datech OSM jsou evidovány vchody pro každou linii, tedy např. stanice „Muzeum“ se tam nachází dvakrát. Pro rozdělení do oblastí to opět nehraje roli, a proto se uloží pouze jedna ze souřadnic. Důvodem k takovému postupu je, že bod na mapě zobrazuje celou stanici metra a ne její vchod.

3.2.2 Oblast Praha

Při sestavování dotazů pro Overpass API bylo potřeba specifikovat oblast, ve které se mají data vyhledávat. Původně jsem dotazování nastavil pouze na oblast Prahy, což fungovalo bez problémů pro stanice metra a lékárny. Tyto objekty jsou v OpenStreetMap jednoznačně přiřazeny k městu Praha v České republice.

Problém však nastává u autobusových zastávek. Ukázalo se, že dotazování pouze na oblast „Praha“ může vést k chybně získaným souřadnicím, protože existuje také vesnička s názvem Praha na Slovensku (pro ověření souřadnice: 48.3629287 / 19.508079 nebo severozápadě od Lučence).

Abych problému předešel, rozšířil jsem dotaz tak, aby nejen vyhledával v oblasti „Praha“, ale zároveň se omezil pouze na území České republiky. Původní dotaz pouze pro Prahu: (*area["name"="Praha"]->.searchArea;*), pro hledání v oblasti České republiky je nutné přidat

tento řádek: (*area["ISO3166-1"="CZ"]->.cz;*). Díky této úpravě se vyhledávání omezuje pouze na Prahu v České republice, což eliminuje možnost získání nesprávných dat z jiné oblasti.

3.2.3 Přepočet souřadnic

Jakmile jsou získána potřebná data z Overpass API, je nutné provést jejich přepočet na souřadnice odpovídající mapě vykreslené v aplikaci. OpenStreetMap pracuje s geografickými souřadnicemi (zeměpisná šířka a délka), zatímco mapa v aplikaci je založena na dlaždicovém systému OSM, kde jsou souřadnice transformovány na X-Y souřadnice odpovídající pixelům na obrazovce.

Řešení:

Pro řešení se použije podobný vzorec jako pro výpočet částí mapy.

$$x = \left(\frac{lon+180}{360}\right) \cdot (Rozměr\ čtverce) \cdot 2^{přiblížení}$$

$$y = (0.5 - \frac{\ln(\tan(\frac{\pi}{4} + \frac{lat \cdot \pi}{360}))}{2\pi}) \cdot (Rozměr\ čtverce) \cdot 2^{přiblížení}$$

Rovnice 2 Převod souřadnic na souřadnice aplikace [16]

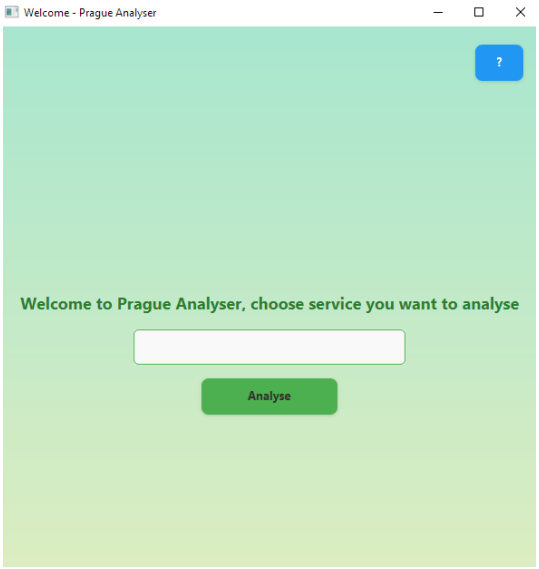
Pomocí tohoto vzorce vypočítáme souřadnice pro nulový bod, což odpovídá souřadnicím levého horního rohu mapy. Tento bod slouží jako určující bod pro následné výpočty. Jakmile máme určeny souřadnice tohoto bodu, provedeme převod zbývajících souřadnic tak pomocí stejného vzorce.

Aby bylo možné správně zobrazit jednotlivé body v aplikaci, je nutné je přepočítat do souřadnic odpovídajícím pixelům na mapě. Toho dosáhneme tak, že od každé vypočítané souřadnice odečteme souřadnice počátečního bodu (nulového bodu). Výsledná hodnota poté určuje polohu daného bodu vůči levému hornímu rohu mapy.

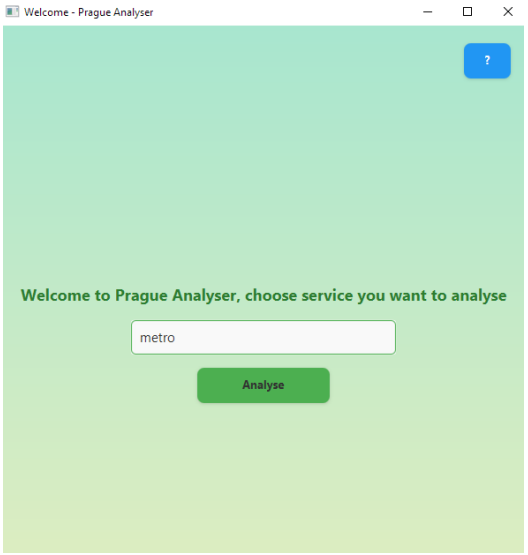
Protože souřadnice pixelů musí být celočíselné hodnoty, vezmeme z výsledného výpočtu dolní celou část. Tento krok zajistí, že získaná hodnota odpovídá přibližné poloze bodu v rámci obrazu mapy a umožní jeho správné umístění v aplikaci. Přibližné proto, že zeměpisné souřadnice počátečního bodu můžou být mírně nepřesné.

4 APLIKACE

Finální aplikace obsahuje pouze dvě scény, což tvoří jednoduché a přehledné ovládání pro uživatele. První scéna slouží jako uvítací, ve které je uživatel přivítán a vyzván k výběru kategorie, kterou může sám zadat podle manuálu v aplikaci (příklad na vyzkoušení: nwr["amenity"="theatre"](area.searchArea);).



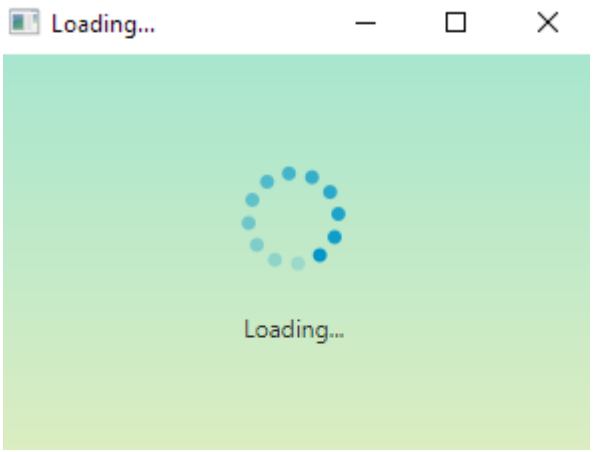
Obrázek 3 Úvodní scéna



Obrázek 4 V úvodní scéně je přednastavená služba pro stanice metra

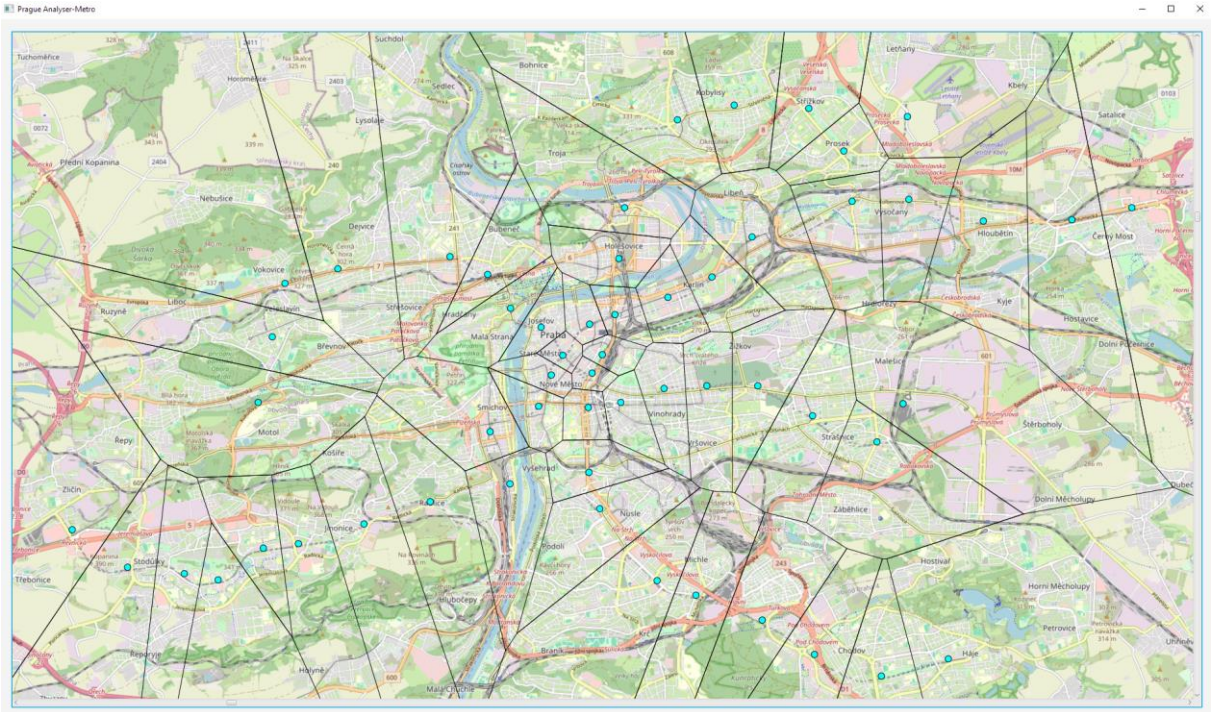
Jakmile zvolí potřebnou službu, klikne na tlačítko „Analyse“, čímž se spustí proces načítání aplikace. V této fázi se začnou zpracovávat data získané z OSM a generovat Voroného diagram na základě zpracovaných dat vybrané služby. Uživatel v tuto chvíli vidí pouze načítací

obrazovku. Po chvíli se rotující kolečko zastaví. Zhruba po třech sekundách se zjeví okno aplikace s mapou, po které se uživatel může pohybovat.



Obrázek 5 Načítaní

Druhá scéna je hlavní částí aplikace, protože vizualizuje rozdělení Prahy na oblasti pomocí Voroného diagramu. Uživatel zde vidí mapu, po které se může libovolně pohybovat, čímž pouze posouvá pohled na mapu. Modrozeleně zvýrazněné body ve tvaru koleček znázorňují polohu vybrané služby. To znamená, že každý z těchto bodů představuje konkrétní lékárnu, stanici metra nebo autobusovou zastávku, což závisí na uživateli, co si zvolil v úvodní scéně.



Obrázek 6 Mapa Prahy rozdělena podle stanic metra

Pro zobrazení digramů tvoří hranici mezi oblastmi černá čára. Každá oblast odpovídá konkrétnímu bodu.

Aplikace navíc obsahuje pár prvků, díky kterým je aplikace uživatelský přívětivější. Na jeden z prvků uživatel narazí, když najede kurzorem myši na některý ze zvýrazněných bodů, poté se zobrazí přidané informace o daném místě. Tyto informace se prozatím týkají názvu služby např. zastávky.

Obrázek 7 Kurzor na bodu služby

Jako další prvek může uživatel spatřit po kliknutí pravého tlačítka na více místech. Po třech a více kliknutí se uživateli zobrazí mnohoúhelník. Každé nové stisknutí pravého tlačítka na myši znamená, setřídění bodů mnohoúhelníku podle úhlu, který svírají s osou x, dále se nalezne konvexní obal pomocí Grahamova skenovacího algoritmu[19] a přepočítá se obsah.

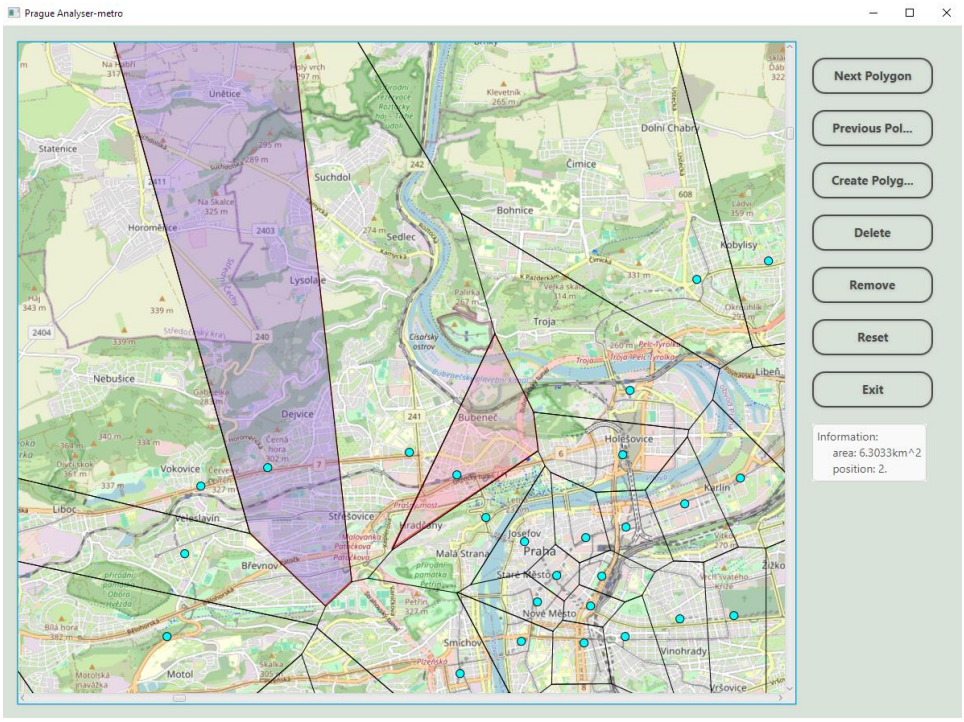
Pro výpočet obsahu jsem použil univerzální vzorec, který pracuje se souřadnicemi kartézské soustavy.

$$S = \frac{1}{2}\left|\sum_{1}^{n-1}(x_i y_{i+1} + x_{i+1} y_i) + (x_n y_1 - x_1 y_n)\right|$$

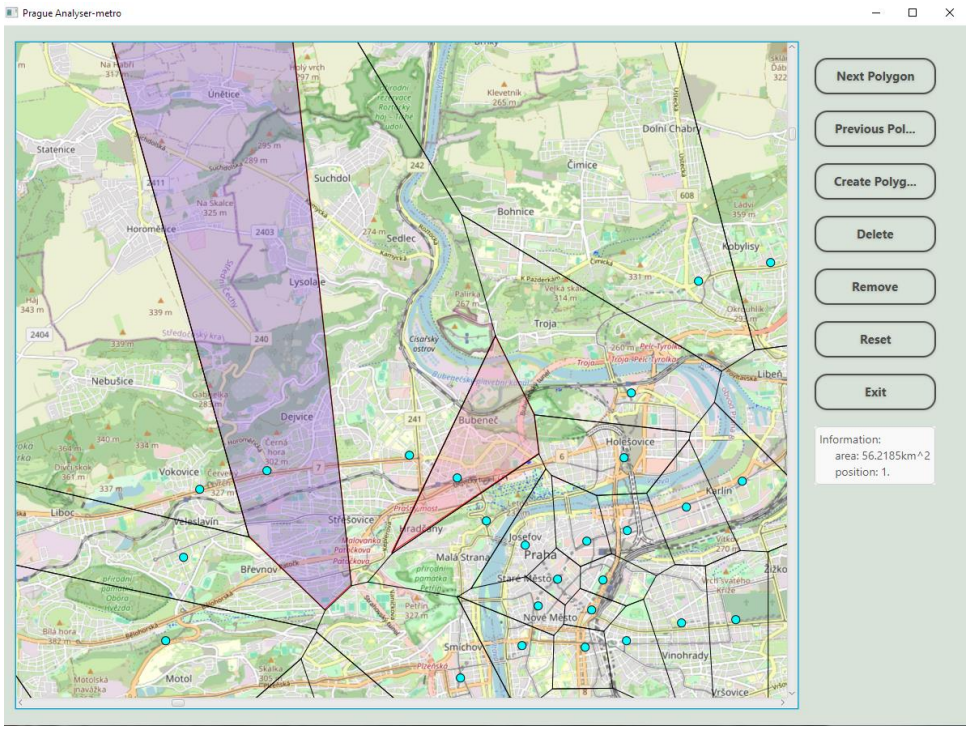
Rovnice 3 Vzorec pro výpočet obsahu mnohoúhelníku[20]

Aby výsledek odpovídal přibližně skutečné rozloze, je nutno vzít odpovídající velikost jednoho pixelu v metrech (19, 109[21]), převést ji na kilometry a umocnit na druhou. Touto hodnotou vynásobíme obsah spočítaný viz rovnice 3.

Výslednou rozlohu odpovídající skutečné může uživatel zahlédnout pod tlačítky po najetí kurzoru na jeden z mnohoúhelníků. Jako další informaci k mnohoúhelníku je pořadí, tedy kolikátý je oproti ostatním vytvořeným mnohoúhelníkům podle obsahu.



Obrázek 8 Aplikace s mnohoúhelníkem (kurzor na červené)



Obrázek 9 Aplikace s mnohoúhelníkem (kurzor na modré)

5 ZÁVĚR

Jak bylo zmíněno v úvodu, práce se zaměřovala na praktické využití Voroného diagramů pro analýzu mapy, čímž se vizualizovala dostupnost veřejných služeb jako jsou lékárny, stanice metra a autobusové zastávky v Praze. Cílem bylo vytvořit aplikaci, která umožňuje nalézt rozdíly v dostupnosti služeb v různých částech města. Výsledkem je mapa, která uživateli poskytuje přehled o tom, jak jsou služby rozmístěny, a pomáhá odhalit oblasti s horší dostupností.

Díky implementovaným řešením lze tuto aplikaci využít nejen pro osobní potřebu uživatelů, ale především k optimalizaci zajištění městské infrastruktury. Může třeba pomoci městským plánovačům lépe pochopit rozmístění veřejných služeb a identifikovat lokality, kde by bylo vhodné vybudovat nové objekty, jako jsou další lékárny nebo zastávky hromadné dopravy. Tato forma vizualizace může být dalším krokem v rozhodování o vybudování nové městské infrastruktury. Přestože projekt nepracuje s nejkratšími cestami vedoucími k jednotlivým bodům ani nezohledňuje faktory, jako je převýšení terénu či reálná doba dojezdu, stále poskytuje cenný nástroj pro analýzu dostupnosti služeb.

Celkově hodnotím projekt jako úspěšný, protože jsem se nejen naučil pracovat se složitým geometrickým algoritmem, díky němuž jsem dokázal efektivně implementovat Voroného diagram do aplikace a vizualizovat jeho výsledky. Práce na tomto projektu mi umožnila seznámit se s aplikováním algoritmů v reálném světě. Zároveň se mi podařilo implementovat Open Source projekt do mé aplikace, což výrazně zlepšilo a zjednodušilo zobrazení mapy v mé aplikaci.

V budoucnu by bylo možné aplikaci rozšířit o další funkce vhodné k analýze i nebo o další města. Dále lze aplikaci graficky vylepšit a tím zajistit, že bude uživatelsky přívětivější.

6 POUŽITÁ LITERATURA

[1] Adamova, Jiří z. wikiwand. [Online] 1. 2 2023. [Citace: 2025.] https://www.wikiwand.com/cs/articles/Georgij_Feodosjevi%C4%8D_Voronoj.

[2] Eppstein, David. wikipedia. [Online] 3. 2 2025. [Citace: 2025.] https://en.wikipedia.org/wiki/Voronoi_diagram.

[3] Felkel, Petr. Voronoi Diagram. 12. 11 2019, [Citace: 2025.] stránky https://cw.fel.cvut.cz/b201/_media/courses/cg/lectures/07-voronoi-ii.pdf.

[4] Folvarčný, Ondřej. muni. [Online] 4. 6 2009. https://is.muni.cz/th/211164/prif_b/BP.pdf.

[5] Samková, Kateřina. VORONEHO DIAGRAMY A JEJICH APLIKACE. *zcu.* [Online] 16. 2 2007. [Citace: 2025.] https://home.zcu.cz/~mika/MM/Galerie%20studentskych%20praci%20MM/2007/Samkov%c3%a1-%20Voroneho%20diagramy%20a%20jejich%20aplikace.pdf.

[6] cadforum. [Online] 2. 2 2024. https://www.cadforum.cz/cz/voroneho-diagramy-v-autocadu-voronoi-diagram-tip14016.

[7] Jlwoodwa. Wikipedia. [Online] 15. 9 2024. [Citace: 2025.] https://en.wikipedia.org/wiki/Fortune's_algorithm.

[8] Kučera, Luděk. Algovize. [Online] 2022. [Citace: 2024-2025.] https://www.algovision.org/Algovision/pool.html.

[9] Kuckir, Ivan. Ivankuv blok. [Online] 8. 3 2011. [Citace: 2025.] https://ivankuckir.blogspot.com/2011/03/fortunuv-algoritmus-jeho-implementace.html.

[10] Zheg, Serena. github. [Online] 5. 4 2016. [Citace: 2025.] https://github.com/serenaz/voronoi/tree/master.

[11] týpek, Takovej normální. Wikipedie. [Online] 22. 9 2024. [Citace: 2025.] https://cs.wikipedia.org/wiki/OpenStreetMap.

[12] OSM. openstreetmap. [Online] [Citace: 2025.] https://www.openstreetmap.org/about/api/.

[13] OpenstreetmapCZ. [Online] 2015-2025. [Citace: 2025.] https://openstreetmap.cz/vyuziti.

[14] OSM, contributors. overpass-turbo. [Online] [Citace: 2025.] https://overpass-turbo.eu/.

[15] wiki. [Online] 26. 11 2024. [Citace: 2025.] https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames.

[16] Dissident. wikipedia. [Online] 1. 9 2024. [Citace:2025.] https://en.wikipedia.org/wiki/Web_Mercator_projection.

[17] Koushanfar, Farinaz. researchgate. [Online] 1 2005. [Citace: 2. 3 2025.] https://www.researchgate.net/figure/oronoi-diagram-example_fig1_220466232.

[18] unaryheap.blogspot. [Online] 3. 7 2015. [Citace: 2. 3 2025.] http://unaryheap.blogspot.com/2015/07/triangulation-fortunes-algorithm-details.html.

[19] geekforgeeks. *geekforgeeks.* [Online] 15. únor 2025. https://www.geeksforgeeks.org/convex-hull-using-graham-scan/.

[20] V., David. wikipedie. *wikipedie.* [Online] 29. 3 2023. https://cs.wikipedia.org/wiki/Mnoho%C3%BAheln%C3%ADk.

[21] OSM. Zoom levels. *wiki.openstreetmap.org.* [Online] 2. 1 2025. https://wiki.openstreetmap.org/wiki/Zoom_levels.

7 SEZNAM OBRÁZKŮ, ROVNIC A UKÁZEK