VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

IMPLEMENTACE ALGORITMU PRO SHLUKOVÁNÍ HRAN GRAFU

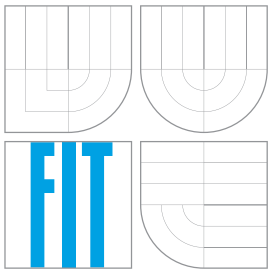
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IVETA KLIMČÍKOVÁ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

IMPLEMENTACE ALGORITMU PRO SHLUKOVÁNÍ HRAN GRAFU

IMPLEMENTING EDGE CLUSTERING FOR GRAPHS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IVETA KLIMČÍKOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2015

Abstrakt

Cílem této práce je prozkoumat možnosti rozložení grafu a shlukování hran, aby se vylepšila celková přehlednost grafu a zredukovalo se rušení. Po shrnutí dostupných nástrojů je detailněji popsána vybraná metoda, která vylepšuje zobrazení grafu bez nutnosti změny pozice uzlů. Práce popisuje implementaci knihovny v jazyce C++ a vytvoření aplikace, pomocí které je možné použít tento algoritmus na jednoduché, ale i rozsáhlejší grafy s větším počtem uzlů a hran. Výsledné grafy je možné exportovat do vektorového formátu SVG a případně zkonvertovat do bitmapového formátu PNG.

Abstract

The objective of the thesis is to explore graph layout and edge clustering to improve graph visibility and the overall edge crossings. A summary of tools focusing on improving of graph visualisation is given. The thesis describes in more details a method of geometry–based edge clustering. Further, the method is implemented in a C++ library. The library itself can handle both simple and more complex graphs with a lot of vertices and edges.

Klíčová slova

Graf, rozvržení uzlů, rozptýlení hran, shlukování hran, kontrolní síť, zobrazení grafu.

Keywords

Graph, node layout, edge dispersing, edge clustering, control mesh, graph visualization.

Citace

Iveta Klimčíková: Implementace algoritmu pro shlukování hran grafu, diplomová práce, Brno, FIT VUT v Brně, 2015

Implementace algoritmu pro shlukování hran grafu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D.

.....
Iveta Klimčíková
27. mája 2015

Poděkování

Týmto by som sa rada poďakovala vedúcemu svojej práce, doktorovi Aleši Smrčkovi, za ochotu a trpezlivosť na konzultáciách a pri tvorbe práce.

Obsah

1 Úvod **3**
1.1 Vlastný prínos práce . . . 3

2 Rozloženie grafu **5**
2.1 Techniky založené na rozmiestnení uzlov . . . 5
2.2 Rozptýlenie hrán . . . 5

3 Zhlukovanie hrán **9**
3.1 Flow map . . . 9
3.2 Hierarchical edge bundles . . . 11
3.3 Confluent drawing . . . 13
3.4 Zhlukovanie hrán s využitím triangulácie Delaunay . . . 14

4 Geometrické zhlukovanie hrán **16**
4.1 Generovanie kontrolnej siete . . . 16
4.2 Triangulácia Delaunay . . . 19
4.3 Zlúčenie hrán a lokálne vyhladenie . . . 20
4.4 Vizualizačné techniky . . . 21

5 Návrh a implementácia **23**
5.1 Vstupné dáta . . . 23
5.2 Vytvorenie kontrolnej siete . . . 25
5.2.1 Generovanie mriežky . . . 26
5.2.2 Vytvorenie trojuholníkovej siete . . . 28
5.3 Zlúčenie hrán grafu . . . 29
5.4 Výstup . . . 29
5.4.1 Bézierove krivky . . . 30
5.4.2 Export grafu . . . 30
5.5 Užívateľské rozhranie . . . 31
5.6 Jednotkové testy kódu . . . 32
5.7 Zhrnutie . . . 33

6 Testovanie a dosiahnuté výsledky **35**
6.1 Vplyv viacerých parametrov pre automatickú tvorbe siete . . . 35
6.2 Rozdiely pri použití mriežky vytvorenej ručne . . . 37
6.3 Ďalšie ukážky testovacích grafov . . . 40
6.4 Rýchlosť vykonania algoritmu . . . 42
6.5 Zhrnutie . . . 43

7 Záver **44**

A Obsah CD **49**

Kapitola 1

Úvod

Grafy sa využívajú v mnohých oblastiach na znázornenie vzťahov medzi dátami. Množstvo vrcholov a hrán obsahujú grafy znázorňujúce trasy leteckých liniek, cesty medzi telekomunikačnými či dopravnými uzlami. S ich narastajúcim počtom výrazne pribúda vizuálne rušenie pri zobrazovaní týchto grafov. Existuje množstvo publikácii venujúcich sa problematike vhodného rozloženia grafu, rozptýlenia či zlučovania hrán.

Prvým krokom tejto práce bolo preštudovať a popísať dostupné algoritmy pre vylepšenia zobrazenia grafov, niektorým zaujímavým metódam sa venujem v prvých dvoch častiach práce. V kapitole 2 sú popísane možnosti zmeny rozloženia celkovej štruktúry grafu, metódy, ktoré vhodne zmenia pozície uzlov, a metódy ktoré umožnia preskúmať lokálne časti grafu. Ak je však poloha uzlov predom pevne daná a pozícia uzlov má nejaký sémantický význam, nie je možné tieto algoritmy použiť.

V kapitole 3 je popísaných niekoľko dostupných algoritmov pre zlučovanie hrán grafu. Pomocou týchto metód je možné dosiahnuť lepšie výsledky, keďže udržujú pôvodnú štruktúru grafu a uzlami nehýbu. Bohužiaľ, väčšina metód je obmedzená na konkrétny typ grafov a len málo z nich spracuje obecný graf.

Nevýhody viacerých spomenutých metód redukuje algoritmus na geometrické zlučovanie hrán, navrhnutý v článku [17]. Kapitola 4 obsahuje teoretický popis krokov tohto algoritmu, ktorý bude použitý ako základ pre vytváranú aplikáciu.

1.1 Vlastný prínos práce

Hlavným cieľom tejto práce bolo vytvoriť knižnicu v jazyku C++ pre algoritmus zhlukovania hrán grafu. Detaily implementácie a popis jednotlivých krokov samotného algoritmu sa nachádza v kapitole 5. Postupne tu popisujem zadávanie a načítanie vstupných grafov v špecifikovanom formáte XML súboru, výstupom je obrázok vo vektorovom formáte. Aby bolo možné jednoducho prekonvertovať SVG súbor na rastrový obrázok typu PNG, k práci je priložený bash script s touto funkcionalitou.

Súčasťou práce sú dve vytvorené aplikácie, ktoré demonštrujú používanie jednotlivých naimplementovaných funkcií. Prvou z možností na vytvorenie upraveného grafu je využitie konzolovej aplikácie, pre overenie korektnosti jednotlivých implementovaných metód sú taktiež vytvorené automatické jednotkové testy. Viacero možností, ako napríklad zobrazenie vstupného grafu, či možnosť manuálne ovplyvniť algoritmus ponúka aplikácia využívajúca Qt toolkit s jednoduchým užívateľským rozhraním.

Na záver sú v kapitole 6 prezentované ukážky s dosiahnutými výsledkami. Taktiež sú

tu rozoberané jednotlivé parametre, ktoré algoritmus priamo ovplyvňujú, a diskutované možnosti, ako sa v niektorých prípadoch dajú výsledné grafy ešte vylepšiť.

Kapitola 2

Rozloženie grafu

Aby sa zredukoval vizuálny šum grafov, bolo navrhnutých viacero metód ako vylepšiť ich priestorové rozvrhnutie. Tieto metódy môžu byť rozdelené do dvoch hlavných kategórii – upravenie pozícii uzlov a vylepšenie rozloženia, rozptýlenie hrán. Problém využitia týchto algoritmov nastáva v prípade, ak je poloha uzlov pevne daná a má pre graf nejaký význam. V tomto prípade je lepšie upraviť graf tak, že zlúčime niektoré hrany a tým dosiahneme menšie vizuálne rušenie. Príkladom týchto metód sú vzorkovanie, zhlukovanie, filtrovanie a animácia.

Jednou z možností, ako zredukovať zmätok pri vykreslení, je filtrovanie menej „dôležitých“ hrán, a tým odhaliť len podstatné vzťahy v grafe [29]. Jednoducho odstránime niektoré hrany, a vo finálnom grafe uchováme len tie, ktoré nesú dôležitejšie informácie. Táto metóda však funguje len v prípade, ak existuje nejaká funkcia, ktorou dokážeme podstatné a nepodstatné hrany rozlíšiť. Ďalším z problémov je uchovávanie celkového kontextu – síce vidíme niektoré čiary, avšak stratili sme informácie o tom, ako boli spojené s ostatnými, teraz neviditeľnými hranami. Podobným pokusom o efektívne využitie filtrovania je metóda, v ktorej sa odstráni väčšina z celkovej dĺžky hrán, a zachová sa len ich malá časť pri krajných bodoch. Pomocou toho je dané, že hrana existovala a vidíme jej smer. Avšak vo finálnom výsledku je príliš zložité zistiť spojenia a vzťahy medzi uzlami.

2.1 Techniky založené na rozmiestnení uzlov

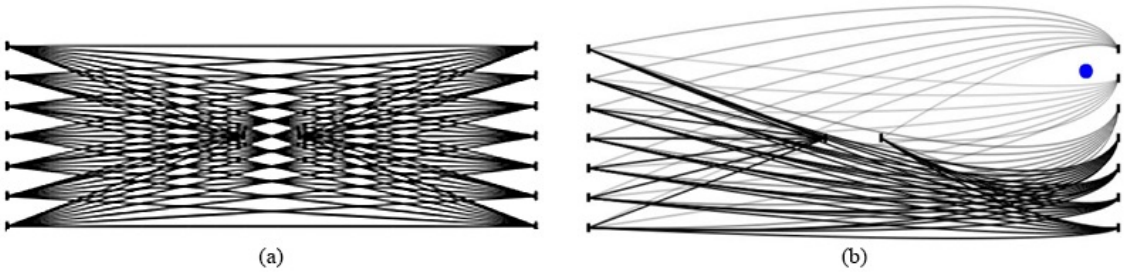
Pre menšie alebo stredne veľké grafy je možné použiť metódy, ktoré vhodným spôsobom zmenia pozície uzlov. Zmenšíme tak počet krížení hrán a celkovo tak vylepšíme vizuálnu prehľadnosť a rozvrhnutie grafu. Avšak, pre husté grafy s väčším počtom hrán, nemusí presunutie uzlov stačiť na to, aby sme dosiahli požadovaný vizuálny výsledok.

Jedným zo spôsobov sú metódy „založené na sile“ (force-based method). V tomto prípade uvažujeme o grafe ako o fyzickom systéme, v ktorom sú uzly modelované ako pevné telesá, a hrany sa považujú za elastické pružiny. Podľa rozličných estetických kritérií alebo špeciálnych požiadaviek môžeme navrhnúť vhodný silový model. Pre väčšie a hustejšie grafy môže byť vytvorenie vhodného modelu náročnejšie.

2.2 Rozptýlenie hrán

Ďalším z prístupov, ako zredukovať zhluky hrán a vylepšiť tak zobrazenie grafu, je rozptýlenie hrán. V jednotlivých lokálnych miestach grafu sú hrany rozptýlené, aby sme mohli

zobraziť základnú štruktúru grafu a odhaliť skryté informácie. Takýto prístup využíva aj technika popísaná v článku [29], nazvaná EdgeLens. Finálny výsledok je znázornený na obrázku 2.1.



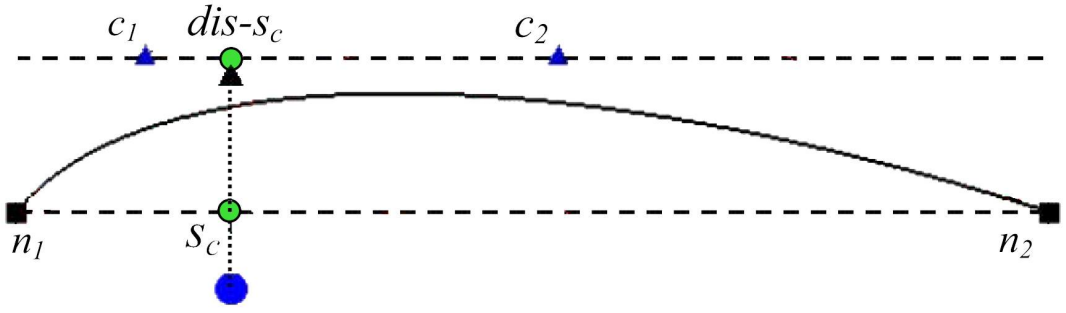
Obrázok 2.1: Vľavo vstupný graf, vpravo upravený pomocou techniky EdgeLens.

EdgeLens

EdgeLens využíva techniku, ktorú by sme mohli prirovnať k lupe. Pomocou lupy určíme jeden alebo viacero miest v grafe, ktoré sú hranami preplnené. Bez toho, aby sme menili pozíciu uzlov, v lokálnych miestach posúvame hrany s veľkým prekrytím, ale zároveň uchovávame všetky pôvodné spojenia uzlov. Spracovanie jednej hrany je možne vidieť na obrázku 2.2. Základný algoritmus je nasledovný:

1. V spracovávanom grafe určíme oblasť, ktorú chceme pomocou metódy objasniť, zadaním jedného hlavného bodu. Na obrázku 2.2 je toto miesto znázornené modrou bodkou. Následne určíme hrany ktoré budú v ďalších krokoch zmenené – sú to všetky hrany v grafe, na ktoré je možné viesť kolmicu z hlavného bodu. Bod, v ktorom sa čiary pretli, označíme ako kontrolný bod S_c.

2. Vypočítame premiestnenie hrany tak, že využijeme kontrolný bod S_c z predchádzajúceho kroku a nastavíme mu novú pozíciu. Jeho vzdialenosť od aktuálnej pozície môže byť definovaná užívateľom a závisí od toho, na akom veľkom priestore chceme hrany rozptýliť.



Obrázok 2.2: Výpočet kontrolných bodov pre jednu krivku pomocou algoritmu EdgeLens.

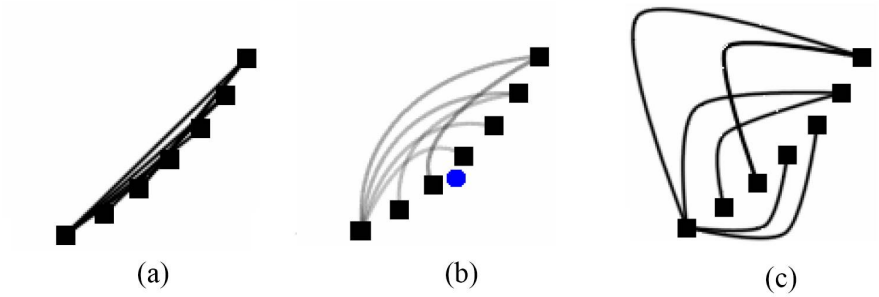
3. Pomocou nového bodu $dis - s_c$ definujeme dva kontrolné body pre vytváranú krivku. Tieto body budú ležať na priamke, ktorá prechádza presunutým bodom a je rovnobežná s pôvodnou hranou. Ich poloha je určená vzdialenosťou bodov S_c a jedného krajného bodu hrany grafu (n_1, n_2), vynásobená zadaným parametrom v rozsahu $(0, 1)$. Zmenou tohto čísla meníme pozície kontrolných bodov c_1, c_2, a tým pádom meníme aj finálny smer krivky a jej tvar.

4. Výsledkom sú štyri body, n_1, n_2, c_1, c_2 potrebné k vykresleniu kubickej Bezierovej krivky, ktorá v grafe nahradí pôvodnú hranu. Jednotlivé kroky opakuje pre všetky určené hrany z bodu 1.

Túto techniku je možné vylepšiť zafarbením niektorých hrán, nastaviť im priehľadnosť, alebo v reálnom čase meniť tvar alebo umiestnenie čiar. Taktiež je možné použiť v jednom grafe túto metódu opakovane, a detailnejšie zobraziť viacero častí grafu súčasne.

Edge Plucking

Podobný prístup využíva metóda od rovnakého autora [28], Edge Plucking („šklbanie“ hranami). Vytvorenie algoritmu bolo inšpirované každodenným životom, keď sa chceme podívať von oknom cez žalúzie. Jednoducho stlačíme palcom jednu alebo viacero lamiel, a zobrazíme miesta za nimi. V popisovanom článku to funguje obdobne s grafmi – užívateľ si vyberie určitú oblasť, a potiahne jednu alebo viacero hrán niektorým smerom. Akonáhle kurzor uvoľníme, hrany grafu sa vrátia na svoje pôvodné pozície, taktiež je však možné hrany pripnúť na konkrétne miesto.



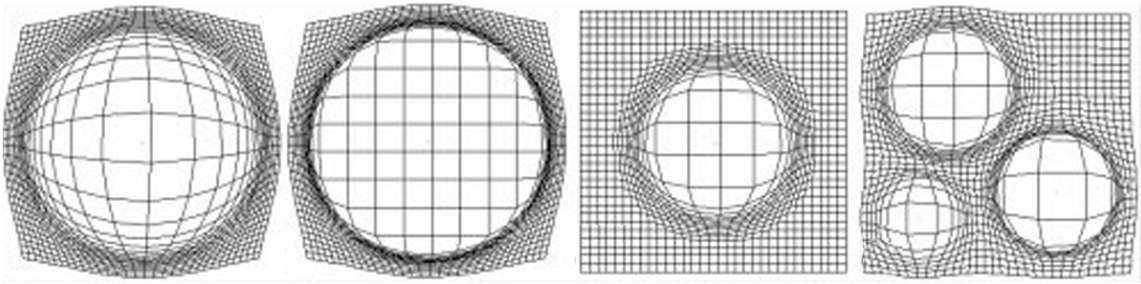
Obrázok 2.3: Porovnanie algoritmov: (a) vstupný graf; (b) metóda EdgeLens; (c) metóda Edge Plucking.

Rovnako ako EdgeLens, aj táto technika posúva hrany bokom. Výhodou Edge Plucking je, že je možné spracovávať jednu hranu po druhej oddelene, ako to vidíme na obrázku 2.3. V prvej časti je zobrazený graf so siedmimi uzlami a niekoľkými hranami. Následne sú na hrany grafu aplikované obe popisované techniky – kým v prvej možnosti je síce graf prehľadnejší než pôvodný, v poslednej časti obrázku je graf zobrazený ešte detailnejšie a priamo vidíme všetky vzťahy medzi uzlami.

FishEye

Ďalšou možnosťou, ako si prezerať rozsiahle grafy, je použiť tzv. FishEye. Pôvodne je FishEye využívané vo fotografiách – existujú objektívy, ktorých šošovka ma veľmi široký uhol záberu a výrazne skreslenie. V tomto prípade je príliš veľké skreslenie zámerom, ako dosiahnuť špeciálny typ snímku. V článku [16] bolo navrhnutých niekoľko techník, ako obdobný princíp využiť pri spracovaní grafov, a zobraziť príliš husté oblasti tak, aby boli zrozumiteľné.

Na obrázku 2.4 vidíme rôzne typy zobrazenia jednoduchého grafu (mriežky). Prvá časť obrázku je základné zobrazenie daného grafu pomocou FishEye, v ktorom skreslené aj okolie grafu, aj zameraná oblasť. Niekedy však je potrebné zobraziť pôvodnú, neupravenú časť grafu, čo môžeme vidieť v druhej časti. Oba typy zobrazenia vytvárajú skreslenie až pri samotných okrajoch grafu. Keď sa však chceme zamerať iba na malú lokálnu oblasť, nie je nutné deformovať väčšinu plochy grafu. Hlavnou výhodou neporušeného okolia je, že túto metódu môžeme použiť na jeden graf opakovane, a preskúmať tak viacero časti grafu a ich detaily (2.4 d).



Obrázok 2.4: Technika FishEye: (a) základné zobrazenie; (b) zameriavaná časť nie je upravená; (c) vybraná len malá lokálna oblasť grafu; (d) metóda FishEye aplikovaná viacnásobne.

Všetky popísané nástroje na rozptýlenie hrán v grafoch sú veľmi užitočné, ak chceme iba preskúmať niektoré zaujímavé časti grafu. Pre rozsiahlejšie grafy s väčšou hustotou hrán však nemusí byť výsledok dostatočný. Je preto vhodné použiť rozptýlenie hrán spoločne s algoritmami, ktoré upravia celkovú štruktúru grafu, čo môže znamenať dosiahnutie ešte lepšieho výsledku.

Kapitola 3

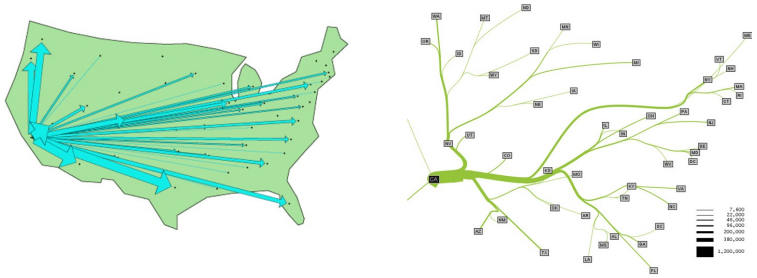
Zhlukovanie hrán

V tejto kapitole bude popísaných niekoľko techník na zhlukovanie hrán. Dôležitou vlastnosťou týchto algoritmov je to, že sa nemenia pozície uzlov. To je veľká výhoda pri upravovaní a zobrazovaní grafov, v ktorých rozloženie uzlov má nejaký význam – napríklad znázornenie trás leteckých liniek, dopravné a komunikačné siete, či mapy rôzneho typu obecne. Bohužiaľ, väčšina z nižšie popísaných techník môže byt aplikovaná len na istý typ grafov, a nie je možné algoritmy zobecniť na všetky grafy.

3.1 Flow map

Flow maps – mapy toku, sú často využívané kartografmi na zobrazenie pohybu objektov z jedného miesta na druhý, ako napríklad počet migrujúcich ľudí či množstvo distribuovaného tovaru. Obdobné grafy môžeme použiť napríklad aj na zobrazenie počtu prenesených paketov v počítačovej sieti. Mapy redukujú vizuálny ruch s využitím kriviek a zlučovania hrán, ktoré majú spoločnú časť trasy či cieľ. Využívajú sa na kreslenie grafov, v ktorom hrany vychádzajú z jedného bodu – hrany zdieľajú tento bod ako koreň stromu. Ak máme navyše k dispozícii informácie o množstve toku medzi miestami, je možné vytvárať cesty stromu široké úmerne k týmto dátam a dostaneme tak presnejšie zobrazenie daného problému.

Avšak väčšina týchto máp je kreslená ručne a existuje len málo dostupných počítačových algoritmov. Metóda navrhnutá v [24] generuje mapy toku s využitím hierarchického zhlukovania, ako vstup je množina uzlov, ich pozície, prípadne spomínané data o množstve toku medzi nimi. Jeden príklad dosiahnutého výsledku je možné vidieť na obrázku 3.1.

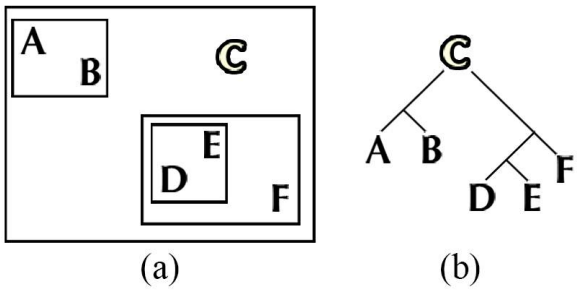


(a) (b)

Obrázok 3.1: Vľavo mapa migrácii, vpravo odpovedajúci vygenerovaný flow map graf.

Prvý krokom je nastavenie rozloženia uzlov. Algoritmus síce mení rozmiestnenie uzlov, ale zachováva ich relatívne pozície vzhľadom k ostatným. Aby bolo možné zobraziť dostatočne široké cesty (krivky v grafe), musí byť vertikálna aj horizontálna vzdialenosť medzi dvoma uzlami väčšia ako maximálna šírka čiary toku, ktorú je možné nastaviť dopredu v pixeloch. Uzly sú usporiadané do dvoch zoznamov, jeden je zoradený podľa súradníc x, druhý podľa y. Uzly potom rozmiestňujeme najprv podľa horizontálnej polohy, následne podľa vertikálnej pozície.

Nasleduje vygenerovanie stromu, ktorý tvorí základ pre finálnu štruktúru mapy. K tomu sa využíva aglomeratívne hierarchické zhlukovanie uzlov [21]. Je to prístup zdola nahor, na začiatku každý zhluk obsahuje práve jeden objekt (uzol). Následne vyberáme dvojicu zhlukov s najmenšou vzdialenosťou, a iteratívne spájame objekty do väčších celkov. Metódu ukončíme po vytvorení hlavného zhluku obsahujúceho všetky uzly. Pri tomto postupe nespájame hlavný uzol z mapy, z ktorého cesty vychádzajú, so žiadnym iným objektom. Po ukončení spájania sme vytvorili štruktúru, z ktorej je možné si predstaviť binárny strom (obrázok 3.2). Koreň stromu je hlavný uzol z mapy, dva menšie zhluky tvoria vždy dva podstromy. Najmenšie zhluky – všetky vstupné body, budú na pozíciách listových uzlov.



Obrázok 3.2: (a) Hierarchické zhlukovanie uzlov; (b) odpovedajúci vygenerovaný binárny strom.

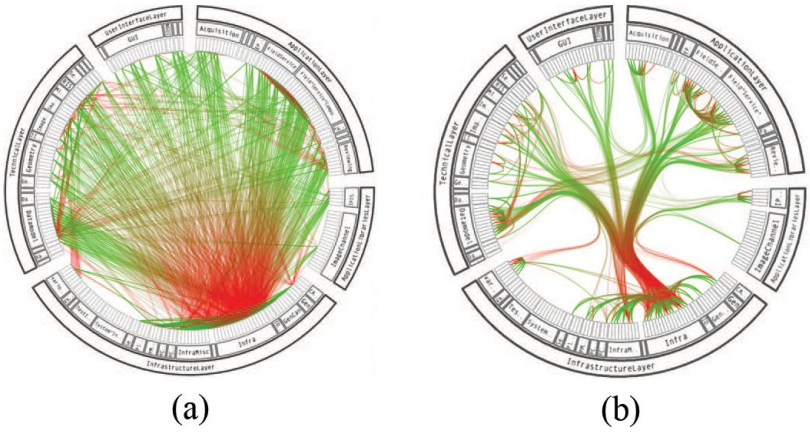
Ďalším krokom je nájdenie uzlov v strome a ich odpovedajúce umiestnenie na mape, ktoré nie sú listové ani koreň – teda akési spojenie ciest z dvoch bodov na mape smerom k rodičovi. K tomu využijeme vytvorenú hierarchickú štruktúru zhlukov z predchádzajúceho bodu. Z rodiča vedieme čiaru smerom do stredu menšieho (synovského) zhluku. Nový bod bude ležať na čiare, v polovici vzdialenosti od rodiča a bodu, v ktorom sa táto čiara pretla s ohraničením zhluku. Rekurzívne generujeme pomocné body, až kým sa nedostaneme k listovým uzlom.

Po týchto krokoch dostaneme základnú štruktúru znázornenia mapy. V článku [24] sú spomenuté ďalšie postupy ako zmenšiť vizuálny šum, presnejšie smerovanie jednotlivých ciest na mape, pridávanie ďalších bodov cez ktoré sú cesty vedené. Aby sme dosiahli požadovane zobrazenie, všetky čiary znázornime pomocou spline kriviek. Ak máme k dispozícii veľkosť pretekajúcich dát medzi uzlami, krivky vykreslíme s rôznou šírkou odpovedajúcou týmto dátam.

Táto metóda bola otestovaná na viacerých zdrojoch – už spomenuté mapy migrácie ľudí či transport tovaru. Dosiahnuté výsledné zobrazenia sa javia ako veľmi sľubné, avšak stále nie je jasné, ako rozšíriť algoritmus aj na obecné grafy, teda všetky grafy v ktorom nie je jasný jeden hlavný (počiatočný) bod.

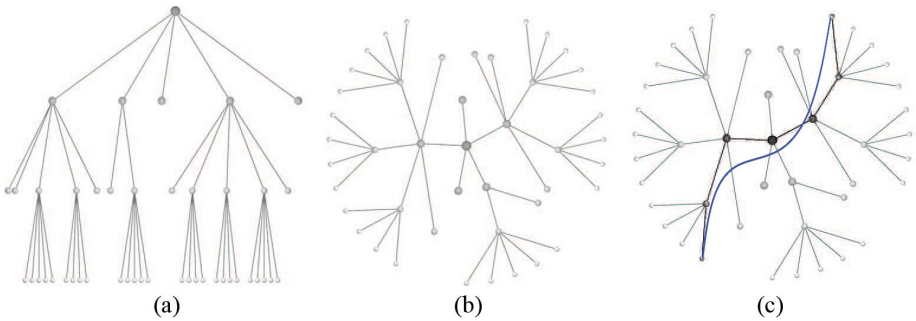
3.2 Hierarchical edge bundles

Efektivitu používania kriviek na redukovanie vizuálneho rušenia demonštruje aj technika v článku [20]. Metóda sa zaoberá zloženými, kombinovanými grafmi (Coupound graph), v ktorých data majú medzi sebou hierarchické vzťahy, tj. vzťah rodič-dieťa, a taktiež aj nehierarchické – susedné vzťahy. Tieto grafy sú často využívané, príkladom je softwarový systém, kde hierarchickú časť tvorí zdrojový kód rozdelený do zložiek, súborov a tried, a susedné vzťahy sú závislosti medzi týmito súbormi.



Obrázok 3.3: Metóda hierarchického zhlukovania hrán: (a) vstupný graf; (b) upravený pomocou kriviek.

Pokiaľ chceme znázorniť iba postupnú závislosť v grafe, je možné využiť viacero dostupných typov znázornenia (obr. 3.4) – jednoduchá stromová štruktúra, hviezdicové rozloženie (Radial tree), balónový strom (Baloon tree), alebo istý typ mapy pre stromy (Treemap). Ak do grafu chceme pridať dodatočné hrany pre susedné vzťahy medzi niektorými uzlami, nastáva väčšinou problém v podobe veľkého vizuálneho ruchu pri zobrazení. Na jeho odstránenie je možné využiť krivky. Aby sme získali smer a trasu krivky každého spojenia dvoch uzlov, využijeme cestu z hrán, ktorou sú tieto uzly spojené v strome. Novú čiaru vedieme popri tejto trase, jednoduchý príklad s využitím hviezdicového rozloženia stromu je možné vidieť na obrázku 3.4 c.



Obrázok 3.4: Typy znázornenia grafu: (a) jednoduchá stromová štruktúra; (b) hviezdicové rozloženie; (c) vytvorenie krivky v grafe.

Ak dve cesty zdieľajú nejakú hranu v pôvodnom strome, zlúčime ich na jeden spoločný segment. Krivky teda počas vytvárania zlučujeme do zväzkov, aby sa ešte viac zredukoval vizuálny šum. Silu zlučovania je možné nastaviť zadaním parametrov do algoritmu, celkový výsledok je tak možné ovplyvňovať. V algoritme [20] sú využívané Beziérové krivky a B-spline krivky s rôznymi nastaveniami zakrivenia. Taktiež je v článku popísaný spôsob zlepšenia ich vykreslenia, nastavenie priehľadnosti, vyhladenie. Dosiahnutý výsledok je zobrazený na obrázku 3.3.

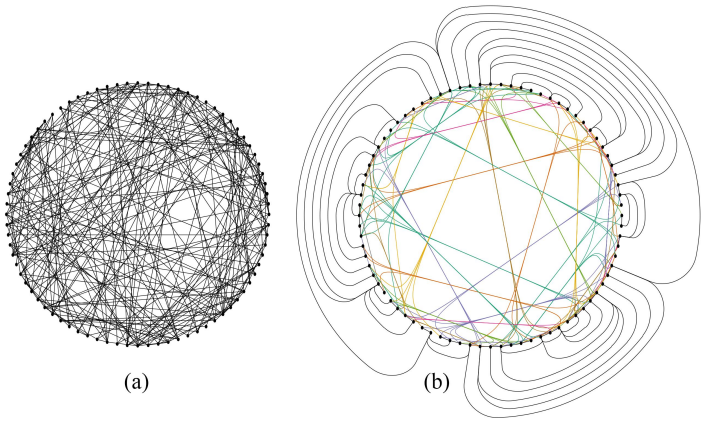
Metóda je zhrnutá ako jednoduchá, flexibilná, dovoľuje užívateľovi vybrať si typ stromového zobrazenia a jemu odpovedajúce vykreslenie spojovacích hrán. Metóda si dobre poradí s veľkým množstvom zobrazovaných hrán a výrazne redukuje ruch. Nevýhodou je opäť nemožnosť algoritmus použiť na všetky typy grafov, a pre nás dôležité obecné grafy bez nutnosti zmeny rozloženia uzlov.

Vylepšenie kruhového rozloženia grafu

V nadväzujúcom článku [19] sa autori snažili ešte viac vylepšiť kruhové rozloženie grafu – teda vykreslenie grafu, v ktorom všetky uzly sú umiestnené po obvode kruhu. Navrhli tri nové, nezávislé techniky ktoré redukujú hustotu zobrazenia hrán a zlepšujú celkovú čitateľnosť grafu.

V prvom algoritme sú uzly grafu presunuté tak, aby sa čo najviac zmenšila dĺžka hrán. Hľadáme teda nové umiestnenie všetkých uzlov tak, aby suma všetkých spojení uzlov v kruhu bola minimálna. V prípade dobrého rozmiestnenia sa výrazne zmenší počet krížení hrán a vykreslenie je prehľadnejšie. Druhá časť popisuje vylepšenie kreslenia grafov tak, že umožňujeme niektorým hranám aby boli vedené aj mimo vnútra kruhu. Výhodou je, že v algoritme je vybraných množstvo krátkych hrán, ktoré je často problematické zobraziť pomocou priamej čiary. Taktiež v tomto prípade nie je nutné meniť rozloženie uzlov. Poslednou navrhovanou metódou je klasické zlučovanie hrán vo vnútri kruhu, rovné čiary sú nahradené krivkami, ktoré navyše zdieľajú spoločné časti trasy.

Najlepší efekt je dosiahnutý pri skombinovaní všetkých troch popísaných metód dohromady. Tieto techniky tak sú schopné zredukovať hustotu hrán a ich kríženie porovnateľne s ostatnými dostupnými metódami. Dosiahnutý výsledok pri využití všetkých techník sa nachádza na obrázku 3.5.



Obrázok 3.5: Kruhové rozloženie grafu: (a) Pôvodný graf; (b) upravený graf.

3.3 Confluent drawing

Planárny je taký graf, ktorého dve rôzne hrany majú spoločné jedine krajné vrcholy, to znamená že tento graf sa dá nakresliť v rovine tak, aby sa žiadne dve hrany nepretínali. Bohužiaľ, väčšina grafov túto vlastnosť nemá. Confluent drawing [18] je technika, ktorá sa zaoberá vizualizáciou grafov, ktoré planárne nie sú. Hlavná myšlienka tohto prístupu je jednoduchá – skupinu hrán zlúčime dohromady, a vykreslíme ich ako zbiehajúce cesty, čo nám umožňuje nakresliť zložitejšie diagramy s pretínajúcimi sa hranami bez kríženia. Samotný algoritmus je možné zjednodušene popísať nasledovne:

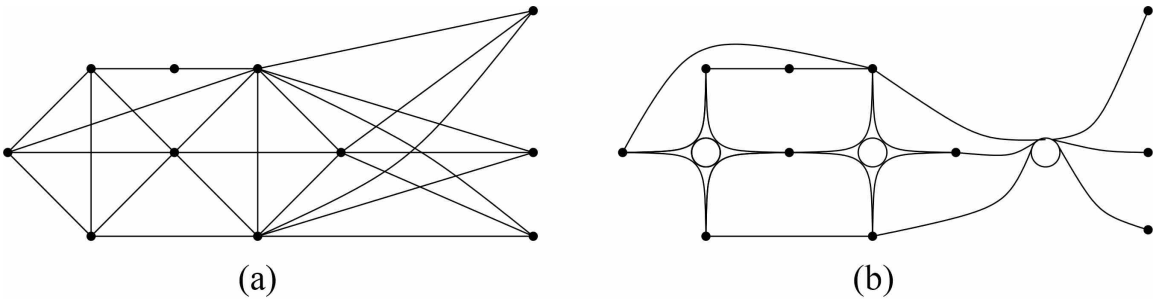
1. Iteratívne nájdeme dve základné štruktúry – úplný graf, kde každé dva vrcholy sú spojené hranou, a úplný bipartitný graf, kde každý uzol z prvej množiny je spojený s každým uzlom z druhej množiny. Uvedené grafy zároveň predstavujú minimálne neplanárne grafy – úplný graf K_5 je neplanárny graf s najmenším počtom vrcholom, a bipartitný $K_{3,3}$ má najmenší počet hrán.

2. Ak graf obsahuje takýto podgraf C, vytvoríme nový vrchol v. Tým, že pridáme vrchol a jeho príslušné hrany do grafu, sa síce zvýši šanca na kríženie hrán, avšak taktiež pribudne viac príležitosti na ich zlúčenie.

3. V pôvodnom grafe odstránime všetky hrany z podgrafu C, a každý jeho uzol spojíme s novo vytvoreným vrcholom v.

4. Nakoniec uzol v nahradíme tzv. „traffic circle“ a dostaneme konečné vykreslenie grafu.

Na obrázku 3.6 je možné vidieť prevedenie menšieho grafu pomocou popisovaného algoritmu.



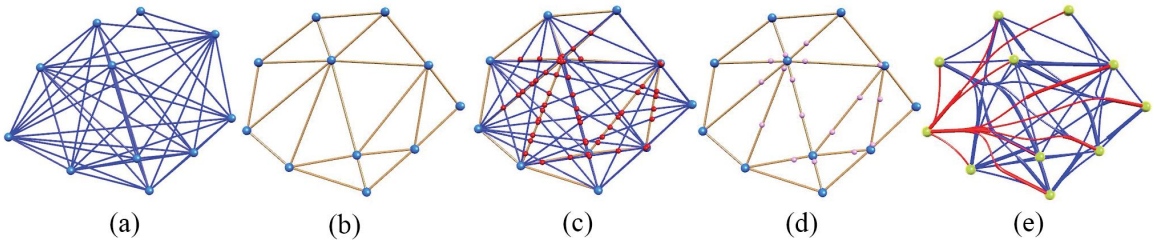
Obrázok 3.6: Confluent drawing: (a) vstupný graf; (b) úprava grafu algoritmom.

Uvedený algoritmus je ľahko implementovateľný pre jednoduché neorientované grafy. Pre zložitejšie, či už orientované, grafy so stromovou štruktúrou a iné, je možné algoritmus modifikovať a dosiahnuť taktiež požadované riešenie. Týmto algoritmom sa článok [18] venuje v ďalších kapitolách. Výsledky algoritmu sú na konci zhrnuté ako vizuálne uspokojivé, grafy sú zobrazené prehľadne a je možné jednoduchšie pochopiť ich štruktúru. Bohužiaľ, nie všetky grafy je možné zobraziť touto technikou, niektoré z nich sú síce spomenuté v závere článku. Pre zložitejšie grafy je však už samotné rozhodovanie, či je na ne možné aplikovať túto metódu, príliš obtiažne.

3.4 Zhlukovanie hrán s využitím triangulácie Delaunay

V článku [25] bol navrhnutý neobvyklý algoritmus pre obecné diagramy. Výhodou navrhnutej metódy je, že zobrazuje vylepšený graf so zachovaním pozícii uzlov, na rozdiel od prístupov, kde sa uzly zhlukujú alebo menia ich pozície. Algoritmus sa dá jednoducho naprogramovať a všetky výpočty môžu byť implementované veľmi efektívne. Navyše, tento prístup ponúka užívateľom flexibilitu pri generovaní finálneho grafu a v priebehu výpočtov povoľuje zadávanie rôznych obmedzení. Vstupom pre algoritmus sú obecné node-link diagramy, čiže klasické grafy s uzlami prepojenými hranami. Pre výpočty musí byť poloha uzlov dopredu známa, a ako už bolo spomenuté, ďalej sa nemení.

Obrázok 3.7 ukazuje ďalej popísané jednotlivé časti algoritmu, prvá časť obrázku zobrazuje vstupný graf. Prvým krokom je výpočet trinagulácie Delaunay z daných vrcholov grafu (obrázok b). Vrcholy sa postupne spájajú po trojiciach do trojuholníkov kým nedostaneme istý typ siete. Algoritmus bude detailne popísaný v ďalších kapitolách.



Obrázok 3.7: Jednotlivé kroky metódy: (a) vstupný graf; (b) trinagulácia Delaunay z vrcholov; (c) priesečníky hrán grafu s Dalaunay linkami; (d) zlúčenie kontrolných bodov; (e) výsledný graf.

Pre každú hranu grafu vypočítame priesečníky s linkami, ktoré sme dostali pomocou triangulácie (červené body na obr. c). Následne z nich určíme jeden alebo viac kontrolných bodov na každej čiare – body je možné zlúčiť pomocou K-means algoritmu (obr. d), prípadne je ich počet a poloha určená užívateľom. Taktiež je možné neskôr tieto body meniť a porovnať tak rôzne dosiahnuté riešenia. Cez tieto body následne prevedieme všetky hrany a vypočítame ich nové smerovanie (obr. 3.7 e).

V tomto prípade sú čiary vykresľované pomocou Nurbs kriviek. Na niektorých miestach kde sa prekrývajú, sú krivky zlúčene dohromady, dosiahneme tak istý flow map efekt. Pri vykresľovaní môžeme začať od základnej úrovne, krivky jednoducho presmerovať tak, aby prechádzali vygenerovanými bodmi. Pre všetky cesty nájdeme kontrolné body cez ktoré prechádzajú, a tým, ktoré zdieľajú rovnakú skupinu uzlov, zlúčime časť ich kriviek dohromady. Celý tento proces môže byť riadený užívateľom – aby nedošlo k zlúčeniu niektorých častí, pridáme jeden alebo viac kontrolných bodov, takže smer a trasa liniek bude viac podobná ich pôvodnému umiestneniu.

Pre niektoré vstupné body je možné dynamicky určovať a meniť tzv. chránenú oblasť okolo nich. Akýkoľvek kontrolný bod potom musí byť v určitej minimálnej vzdialenosti od vybraného uzlu, čo znamená že hustota vykresľovaných kriviek okolo uzlu bude menšia a celkový výsledok nám dodá väčší prehľad v zadaných oblastiach – väčšinou sa jedná o nejakým spôsobom dôležité časti grafu.

Spájanie všetkých liniek nemusí stačiť pre rozsiahlejšie grafy s príliš veľkým počtom uzlov a hrán. V ďalšej časti článku bol navrhnutý spôsob ako vizuálny efekt ešte vylepšiť, a

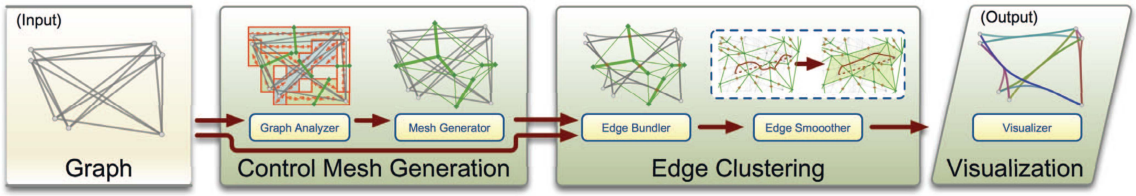
to zlúčením uzlov a odoberaním hrán v samotnej triangulácii. Postupne je možné odoberať najkratšie hrany, tým dostaneme menší počet kontrolných bodov a konečné zobrazenie bude ešte prehľadnejšie.

Popísaná metóda je v závere zhrnutá ako jednoducho implementovateľný algoritmus, ktorý ponúka dostatočne uspokojivý vizuálny efekt pre veľké grafy. Podľa iného zdroja [17] ale metóda vytvára príliš veľa zigzag ciest pre rozsiahlejšie grafy, a tak je pre užívateľov zložitejšie rozoznať smer kriviek a koncové body.

Kapitola 4

Geometrické zhlukovanie hrán

Ako už bolo spomínané, pri znázornení rozsiahlych grafov nadmerné kríženie hrán vytvára veľký vizuálny ruch a je zložité graf preskúmať. V článku [17] je navrhnutý nový algoritmus založený na geometrickom zhlukovaní hrán, v ktorom sa hrany zlučujú do zväzkov, aby sa zredukovalo celkové kríženie hrán. V porovnaní s predchádzajúcimi technikami, táto metóda pracuje s obecnými grafmi a nemení pozíciu uzlov. Jednotlivé kroky algoritmu sú zobrazené na obrázku 4.1 (všetky obrázky v kapitole prevzaté a upravené z [17]).



Obrázok 4.1: Geometrické zhlukovanie hrán – jednotlivé kroky algoritmu.

Prvým krokom je rozdelenie grafu na lokálne časti, pre každú oblasť zanalyzujeme rozloženie čiar a zistíme hlavný smer. Potom vygenerujeme kontrolnú sieť, na ktorej hranách sa budú nachádzať kontrolné body. Sieť môže byť generovaná s rôznym stupňom detailnosti, a taktiež manuálne či automaticky. Následne budeme „nútiť“ všetky čiary aby prechádzali cez tieto body, zväzky hrán sa tak budú tvoriť prirodzene. Aby sme nakoniec zlepšili celkové rozloženie grafu, zavedieme lokálne vyhladenie na všetky zigzag krivky a použijeme pokročilejšie vizualizačné techniky na detailnejšie zobrazenie štruktúry konečného grafu. Užívatelia môžu celkový výsledok ovplyvniť konečným nastavením typu zafarbenia, zvýšením či znížením priehľadnosti.

4.1 Generovanie kontrolnej siete

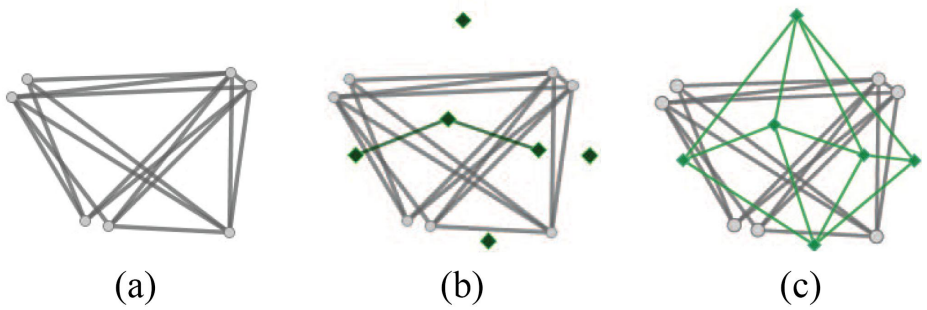
Veľmi dôležitú rolu pri procese zhlukovania hrán a dosiahnutí finálneho rozloženia grafu má kontrolná sieť. Pri dobrom vygenerovaní je možné v grafe eliminovať veľké množstvo križení hrán, zlúčiť hrany s rovnakým smerom a dĺžkou, a minimalizovať vzdialenosti medzi originálnymi priamymi čiarami a finálnymi vykreslenými krivkami.

Jednoduché metódy na generovanie siete, napríklad ak využívame iba pozície uzlov, nemusia byť dostatočné. Príkladom môže byť spájanie uzlov do trojuholníkov pomocou

triangulácie Delaunay, ktorá bude popísaná nižšie. To však nefunguje pre množstvo grafov, pretože neberieme do úvahy celkovú štruktúru grafu a rozloženie hrán. V popisovanom algoritme je prvým krokom detekcia zhlukov hrán, v ktorých sa hrany nachádzajú blízko pri sebe a majú podobný smer. Následne sa vygenerujú hrany siete, ktoré prechádzajú cez dané zhluky.

Manuálne generovanie

Jedným možným riešením je umožniť užívateľovi ručne vytvoriť kontrolnú sieť na základe vstupných dát. Sieť môže byť manuálne vytvorená celá, alebo je možné zadať iba niekoľko vrcholov, alebo priamo definovať hrany siete tak, aby prechádzali cez zhluky hrán grafu. V tomto prípade nasleduje automatické spojenie zadaných vrcholov či hrán, aby sa vytvorila trojuholníková mriežka. Na obrázku 4.2 je možné vidieť kroky manuálne vytvárania kontrolnej siete. Pre jednoduché grafy s jasnými zhlukmi hrán nie je problém kontrolnú sieť vytvoriť a hlavnou výhodou je to, že užívateľ priamo ovplyvňuje celkový výsledok zobrazenia grafu. Avšak pre hustejšie grafy je vizuálne hľadanie zhlukov hrán a zadávanie bodov oveľa zložitejšie a časovo náročnejšie.

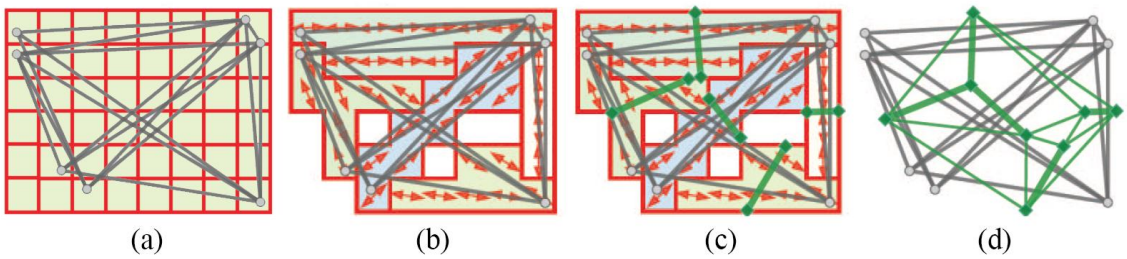


Obrázok 4.2: Generovanie kontrolnej siete manuálne: (a) vstupný graf; (b) zadané kontrolné body a dve hrany siete; (c) vytvorená kontrolná sieť.

Automatické generovanie

Lepším spôsobom získania kontrolnej siete je jej automatické vypočítanie na základe pozícií hrán. Základný postup je znázornený na obrázku 4.3.

V prvom kroku nájdeme pre graf minimálny ohraničujúci obdĺžnik, a túto plochu rozdelíme na pravidelnú mriežku. Veľkosť jednotlivých buniek alebo ich počet môžu byť zadané užívateľom. Pre každú bunku zistíme počet uzlov, ktoré spadajú do danej bunky a počet čiar, ktoré cez ňu prechádzajú. Pre všetky čiary vypočítame vektor príznakov, ktorý udáva ich smer. Následne zistíme, aký uhol (smer) prevažuje v jednotlivých častiach. Ak cez bunku mriežky neprechádzajú žiadne hrany, bude bunka v ďalších krokoch ignorovaná. Následne zlúčime menšie oblasti s vektormi s podobnými smermi do väčších. Bunky spájame dovtedy, až kým maximálny rozdiel uhlov nedosiahne nejaký predom definovaný prah (napríklad 15°). Pre každú takto získanú oblasť vypočítame nový hlavný smer ako vážený priemer vektorov menších častí.



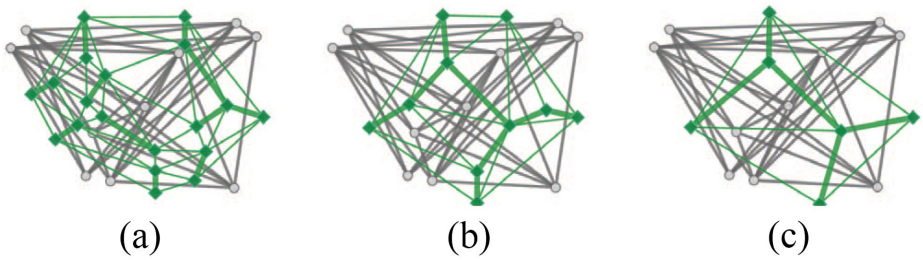
Obrázok 4.3: Generovanie kontrolnej siete automaticky: (a) rozdelenie grafu pomocou mriežky; (b) zlúčenie regiónov a výpočet smerov hrán; (c) vytvorené kolmice cez ťažisko každej oblasti; (d) vytvorená kontrolná sieť.

Hrany generovanej kontrolnej siete prechádzajú ťažiskom každej oblasti a budú kolmé na hlavný smer zhluku. Po spracovaní každej oblasti sme dostali množinu hrán siete. Kontrolné vrcholy, čiže krajné body týchto hrán, ktoré sa nachádzajú blízko seba zlúčime dohromady, prípadne, ak je potrebné, použijeme Poissonovo vzorkovanie na získanie viacerých bodov. Nakoniec pomocou triangulácie Delaunay (kapitola 4.2) dostaneme finálnu trojuholníkovú kontrolnú sieť.

Hierarchické generovanie

Ak si vytvoríme viacero hierarchických kontrolných sietí, môžeme dosiahnuť rôzne úrovne detailnosti zobrazenia grafu. Pri automatickom generovaní umožňujeme zlučovanie buniek v mriežke do väčších oblastí na základe užívateľom definovaného rozdielu uhlov. Viacero úrovni detailov dostaneme tak, že predom špecifikujeme skupinu prahov (obr. 4.4).

Druhou možnosťou je zmena rozlíšenia mriežky. Plocha grafu môže byť rozdelená napríklad na 64 x 64, 128 x 128, či 256 x 256 buniek, čím dostaneme tri rôzne kontrolné siete, ktoré ovplyvňujú finálne rozloženie grafu. Táto flexibilná kontrola nastavenia rôznych stupňov detailnosti zobrazenia je veľkou výhodou popisovaného algoritmu.



Obrázok 4.4: Generovanie kontrolnej siete hierarchicky - rozlíšenie na základe počtu kontrolných uzlov.

4.2 Triangulácia Delaunay

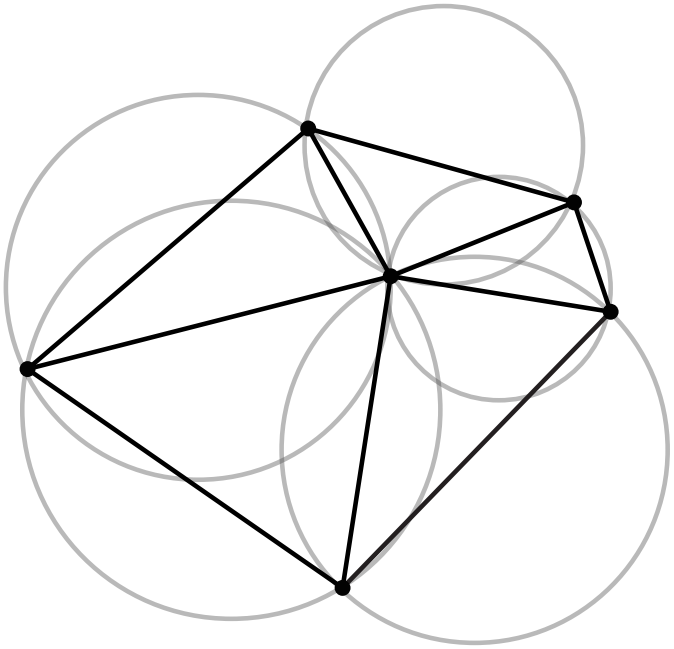
Ako bolo popísané v predchádzajúcej kapitole, jedným z krokov algoritmu je spojenie bodov pomocou triangulácie Delaunay. Triangulácie [14] majú význam vo viacerých oblastiach; najčastejšie sa využívajú v kartografii, napríklad pri tvorbe digitálnych modelov terénov, modelovanie prírodných javov, v biometrii pri detekcii odtlačkou prstov. Veľké využitie majú taktiež aj v počítačovej grafike, pri spracovávaní obrazu – segmentácia, rozpoznávanie vzorov, alebo vizualizácia priestorových dát v scénach.

Triangulácia nad množinou bodov predstavuje také planárne rozdelenie, ktoré vytvorí zoznam trojuholníkov pre ktoré platí:

- Ľubovoľné dva trojuholníky majú najviac jednu spoločnú hranu.
- Zjednotenie všetkých vrcholov trojuholníkov tvorí celú vstupnú množinu bodov.
- Vo vnútri žiadneho trojuholníka neleží žiadny ďalší bod.

Pri implementácií triangulačného algoritmu sa kladie doraz na jednoduchosť, dostatočnú rýchlosť výpočtu, v určitých prípadoch je potrebné previezť dáta do vyšších dimenzií. Existuje viacero typov, základné rozdiely sú napríklad v tvare trojuholníkov, ktoré daný algoritmus produkuje. Niekedy vstupom do algoritmu nie len množina bodov, ale aj povinné hrany, ktoré je nutné zachovať. V tom prípade hovoríme o obmedzenej triangulácii (Constrained Triangulation).

Pre nás dôležitý je algoritmus Delaunay Triangulation [15], ktorý je najčastejšie používaný typ triangulácie. Pôvodná verzia metódy pracovala iba s dátami v dvojdimenzionálnom priestore, dnes ale existujú varianty aj pre trojrozmerný priestor, ktorý je práve často využívaný v počítačovej grafike.

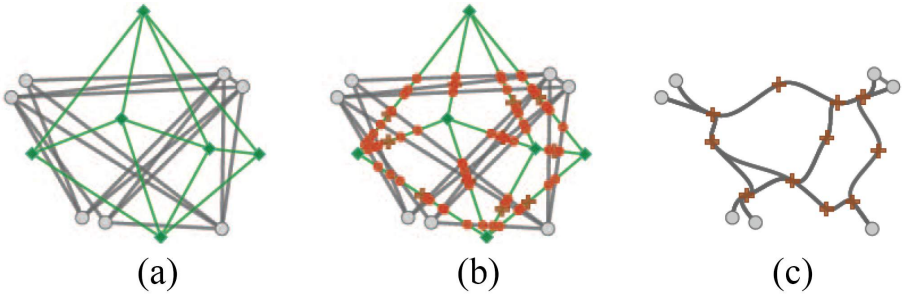


Obrázok 4.5: Triangulácia Delaunay.

Základným princípom algoritmu je hľadanie takej trojice bodov, kde v priestore ohraničenom kružnicou obsahujúcou dané tri body, neleží žiadny iný bod zo vstupnej množiny. V prípade nájdenia takého setu vytvoríme trojuholník, ktorého hrany tvoria úsečky spájajúce tri dvojice vybraných bodov, a trojuholník pridáme do výslednej siete. Algoritmus sa zároveň snaží maximalizovať minimálny uhol – ako vstupný parameter je možné zadať minimálnu hodnotu uhla, ktorú musia spĺňať všetky vnútorné uhly vytváraných trojuholníkov. Na obrázku 4.5 vidíme popisované vytváranie trojuholníkov a pomocných kružníc, vo vnútri ktorých neleží žiadny iný vstupný bod.

4.3 Zlúčenie hrán a lokálne vyhladenie

Ďalším krokom po vygenerovaní kontrolnej siete je získanie hlavných bodov, na základe ktorých budeme zlučovať hrany grafu. Na obrázku 4.6 vidíme graf a odpovedajúcu manuálne vytvorenú sieť. Získame všetky priesečníky medzi pôvodnými hranami grafu a čiarami kontrolnej siete, tieto vrcholy sú zobrazené červenými bodmi (obr. 4.6 b). Pomocou algoritmu K-means vypočítame jeden alebo viacero hlavných bodov pre každú hranu. Následne pôvodné čiary v grafe presmerujeme tak, aby prechádzali cez tieto body, čím automaticky vytvoríme zväzky hrán a tento graf so zlúčenými hranami bude prehľadnejší (obr. 4.6 c). Na vykreslenie nových hrán je možné použiť rôzne typy kriviek.



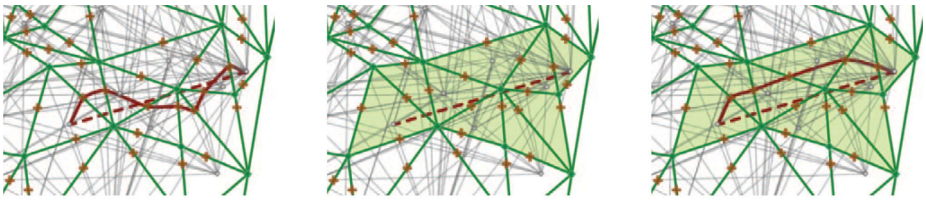
Obrázok 4.6: Zlúčenie hrán: (a) vstupný graf s kontrolnou sieťou; (b) priesečníky pôvodných hrán grafu s hranami kontrolnej siete; (c) vytvorené krivky na základe kontrolných bodov.

K-means je jednoduchý zhlukovací algoritmus, v našom prípade sa bude algoritmus vykonávať zvlášť pre body ležiace na každej hrane v sieti. Na začiatku sa určí počet zhlukov, ktoré chceme dosiahnuť, vytvoríme minimálne jeden na každej z hrán. Pre každý zhluk zvolíme jeden bod ako stred, a ostatné body pridelíme k najbližšiemu z centrálnych bodov. Následne prepočítame stred pre všetky zhluky a algoritmus opakujeme, až kým žiadnemu z vrcholov nebol zmenený priradený zhluk.

Lokálne vyhladenie zigzag ciest

Graf so zlúčenými hranami získaný z predchádzajúceho bodu nemusí byť vizuálne dostatočne uspokojivý, pretože niektoré hrany môžu obsahovať príliš veľa zigzag ciest. Tieto cesty môžu ukazovať nesprávne smerovanie hrán, a celkové pochopenie grafu bude zavadzajúce. Aby sa vyriešil tento problém, bol navrhnutý algoritmus lokálneho vyhladzovania, ktorého hlavné kroky môžeme vidieť na obrázku 4.7.

Každá priama hrana v grafe je po aplikovaní vyššie popísaných bodov vykreslená ako krivka. Preto bola najprv navrhnutá metrika kvality, ktorá udáva to, ako veľmi nová krivka reprezentuje pôvodnú priamu čiaru. Hodnota Q_e pre každú čiaru je vypočítaná na základe údajov o zakrivení krivky, berieme do úvahy počet bodov v ktorých čiara mení smer, a maximálnu Euclidovú vzdialenosť medzi krivkou a originálnou hranou.



Obrázok 4.7: Lokálne vyhladenie: (a) vygenerovaná krivka; (b) určená obmedzujúca oblasť; (c) vytvorenie novej krivky.

Na základe vypočítanej hodnoty môžeme určiť množinu kriviek so slabou kvalitou a aplikovať na ne lokálne vyhladenie. Hlavný cieľ je nájsť inú cestu, alebo množinu nových kontrolných bodov cez ktoré bude pôvodná hrana grafu prechádzať. Prvým krokom je nájdenie lokálnej oblasti na ktorú sa obmedzíme pri hľadaní novej trasy (obr. 4.7 b). Oblasť tvoria všetky trojuholníky v kontrolnej sieti, cez ktorú prechádza spracovávaná hrana, a susedné oblasti ktorých vrcholy ležia v jej blízkej vzdialenosti. Ak je nastavený parameter vzdialenosti vyšší, tým je väčšia šanca že bude nájdená lepšia nová trasa. Po určení tejto oblasti nájdeme všetky možné cesty pre originálnu hranu a vyberieme tú, ktorá ma najlepšiu hodnotu dostanú pomocou metriky kvality (obr. 4.7 c).

4.4 Vizualizačné techniky

V hustých grafoch môže byť aj po aplikovaní metód na zlučovanie hrán stále nedostatočná možnosť detailnejšie zobraziť základné štruktúry. Rozloženie grafu je preto v ďalších krokoch vylepšené pokročilejšími vizualizačnými technikami, ako je zafarbovanie jednotlivých segmentov grafu, nastavenie priehľadnosti, či animácia.

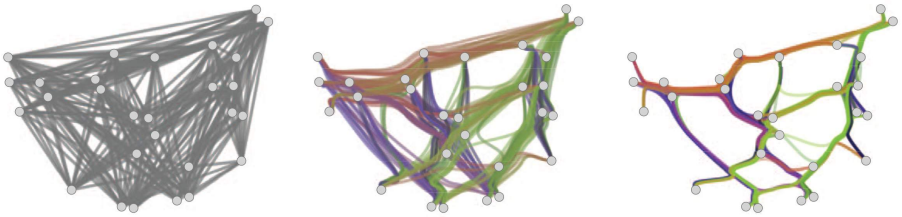
Nastavenie farieb a priesvitnosti

Po zlúčení hrán môžeme vypočítať viacero hodnôt pre krivky v grafoch. Napríklad, každá časť krivky reprezentuje rozličný počet pôvodných hrán v grafe, vieme zistiť rozdiel vzdialenosti nových čiar od základných, alebo môžeme krivky roztriediť podľa toho, aký uhol mali hrany grafu ktoré nahradili. Na základe týchto vlastností môžeme definovať funkciu, ktorá bude určovať typ zafarbenia a silu priehľadnosti čiary. Rozličné farby tak budú mať čiary, ktoré sú krátke, a tie, čo prechádzajú cez celý graf. Prípadne budú iné tie, ktoré majú vodorovný alebo zvislý smer.

Animácia

Posledným vylepšením popísaným v článku [17] je zavedenie rôznych typov animácií. Ak napríklad bude zobrazený celý proces zhlukovania hrán a to, ako sa hrany menili z priamych

čiar na krivky a potom boli zlúčené dohromady, užívateľ dostane lepšiu predstavu o celom grafe a môže jednoduchšie pochopiť niektoré vzory v grafe, ktoré by boli nejasné ak by sa zobrazil až finálny výsledok. Na obrázku 4.8 sú ukážky, ako sa graf mení počas aplikovania algoritmu, zároveň sú hrany rozdielne zafarbené podľa smeru.



(a) (b) (c)

Obrázok 4.8: Animácia algoritmu s využitím zafarbenia hrán podľa smeru.

Kapitola 5

Návrh a implementácia

Základom tejto práce je implementácia knižnice pre jazyk C++, ktorá obsahuje metódy pre algoritmus popísaný v kapitole 4. V tejto kapitole sa nachádza bližší popis jednotlivých implementačných detailov, sú tu diskutované viaceré alternatívne knižnice a formáty, ktoré by v práci bolo možné využiť.

Pre jednoduchú ukážku vytvorenej funkčnosti bolo vytvorené užívateľské rozhranie s využitím knižnice Qt. Ako vstup sme zvolili jednoduchý XML súbor, pomocou ktorého je možné spracovať rozličné grafy definované v nižšie popísanom formáte. Samotný algoritmus, konkrétne automatické vytvorenie kontrolnej siete, je možné užívateľsky ovplyvniť zadaním a zmenou štyroch rozličných parametrov priamo v implementovanom GUI. Druhou možnosťou, ako ovplyvniť výsledný graf, je vytvorenie kontrolnej siete ručne. To je možné jednoduchým zadaním riadiacich bodov, ktoré sú následne programovo spojené do trojuholníkov. V poslednom kroku je možné finálny graf exportovať do vektorového formátu SVG a priamo tak porovnať dosiahnuté výsledky.

Súčasťou práce sú taktiež jednotkové testy na otestovanie funkčnosti jednotlivých funkcií. Bližší popis a štatistiky pokrytia implementovaného kódu sa nachádzajú v kapitole 5.6.

5.1 Vstupné dáta

Prvým krokom pri vykonávaní algoritmu je zadanie vstupných dát grafu. Informácie o uzloch sú uložené a načítavané zo súboru, po dodržaní presného, dolu uvedeného formátu, je možné algoritmus aplikovať na akýkoľvek vytvorený graf.

Z dostupných nástrojov bola možnosť využiť súbor v podobnom formáte, v akom sa zadávajú vstupné dáta pre nástroj na zobrazovanie grafov `Graphviz` [5]. Tento nástroj ponúka širokú škálu využitia – výsledné grafy je možné uložiť ako rastrové obrázky a vektorové zobrazenie typu SVG, exportovať do pdf súborov, či zobraziť interaktívnou formou v prehliadačoch na to určených. Software ponúka viacero funkcií ako vylepšiť konkrétne diagramy, či už rôzne nastavene farieb, typu písma, štýlu čiar, je možné ručne definovať súradnice uzlov či si nechať celkové rozloženie grafu vygenerovať automaticky. Vstupné dáta sú zadané v textovom jazyku `DOT language`:

V ukážke na obrázku 5.1 vidíme ukážkový graf, kde sú zadaný štyri identifikátory uzlov, a definované tri hrany medzi nimi. Pomocou atribútu `pos` je možné určiť súradnice vrcholov, čo je nutná podmienka pre implementovaný algoritmus. Zadávanie dát je jednoduché, avšak z množstva možností, ktoré `Graphviz` ponúka, by bolo využité len minimum.

```
graph GraphName
{
        1 [pos = "10, 10", width = "0.75", height = "0.5"];
        2 [pos = "15, 15"]
        3 [pos = "10, 20"]
        4 [pos = "20,30"]

        1 - - 2 - - 3
        2 - - 4
}
```

Obrázok 5.1: Formát vstupných dát pre nástroj Graphviz.

Aby bol vstupný súbor ešte prehľadnejší a obsahoval iba potrebné základné údaje o uzloch, ich pozície a hrany medzi nimi, bol vytvorený nový formát zadávania grafu vo forme XML. Ukladanie v XML súbore taktiež uľahčí a urýchli načítanie dát do potrebných objektov v programe. V tabuľke 5.3 je uvedená základná štruktúra navrhnutého vstupu. Popisuje jednoduchý graf odpovedajúci hore uvedenej ukážke formátu pre `Graphviz`. Každý uzol, XML element `node`, obsahuje atribúty s jeho identifikátorom `id` a jeho pozície `x` a `y`. Hrany sú taktiež označené pomocou `id`, a navyše obsahujú identifikačné číslo zdrojového a cieľového uzla.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<graph>
        <node id="1" cx="10" cy="10" />
        <node id="2" cx="15" cy="15" />
        <node id="3" cx="10" cy="20" />
        <node id="4" cx="20" cy="30" />

        <edge id="1" source="1" target="2" />
        <edge id="2" source="2" target="3" />
        <edge id="3" source="2" target="4" />
</graph>
```

Obrázok 5.2: Navrhnutý vstupný formát dát.

XML súbor je možné programovo spracovať troma základnými spôsobmi – s využitím štruktúry DOM, push model (SAX), a pull model (StAX) [26]. Všetky tri ponúka napríklad C++ knižnica Xerces-C+ [6].

DOM

DOM - Document Object Model je multi-platformná a na jazyku nezávislá špecifikácia pre reprezentáciu objektov nie len XML súborov, ale aj pre typ html či xhtml. S využitím DOM parseru sa v pamäti vytvorí stromová štruktúra celého súboru, a v programe sa potom iteruje cez jednotlivé elementy. Prechádzanie dát je jednoduchšie, avšak nevýhodou je veľká pamäťová náročnosť, a využitie tohto spôsobu tak nie je vhodné pre rozsiahle súbory s príliš

veľkým počtom uzlov. Práve prácu s pamäťou vylepšuje algoritmus SAX.

SAX

SAX - Simple API for XML nenačítava do pamäte celý dokument, ale prechádza XML súbor riadok po riadku. Pri narazení na rôzne časti elementov, ako sú otvárací a uzatvárací tag, komentáre či samotné dáta, vyvoláva rozličné typy udalosti. Po spracovaní konkrétnej časti je zvyčajne „zahodená“ väčšia časť informácií a pokračuje sa na ďalší element, čo je veľký rozdiel práve od DOM prístupu, kde obsah celého súboru v pamäti ostáva po dlhú dobu. Tento prístup je nazývaný taktiež aj push model – algoritmus predáva užívateľovi jednotlivé dáta vyvolaním konkrétnej metódy. Opakom je model pull, StAX prístup.

StAX

StAX (Streaming API for XML) obsahuje kurzor, ktorý predstavuje aktuálny bod v súbore, na ktorom sa algoritmus nachádza. Pomocou neho užívateľ prechádza jednotlivé položky XML dokumentu, a získava tak informácie ktoré potrebuje. Tento prístup bol vyvinutý pôvodne pre jazyk Java, a predstavuje akýsi stred medzi dvoma predchádzajúcimi prístupmi – o prechádzanie dokumentu sa stará samotný spracovávajúci program, ale spotrebuje výrazne menej pamäte než DOM.

V implementácii by bolo možné využiť ktorýkoľvek z prístupov. Predpokladom je, že program nebude pracovať s príliš veľkými XML súbormi (˜MB), a tak nie je problém s používaním DOM parseru. Pri programovaní bol kladený doraz na jednoduchosť implementácie, a bola zvolená voľne dostupná knižnica `pugixml` [22]. Táto knižnica kladie doraz na rýchlosť načítania súboru a využíva práve popisovaný DOM parser.

Pri načítaní súboru v programe je ako koreňový uzol uložený element `graph`. V cykloch sa následne prechádzajú synovské elementy, atribúty `node` a `edge`. Samotné uloženie dát pre uzly a hrany v programe je v objektoch typu `Node` a `Edge`. Celkový graf potom reprezentuje instancia triedy `Graph`, ktorá obsahuje mapu jednotlivých uzlov, kde ako kľúč je použité id uzla, s uložená hodnota je ukazateľ na konkrétny objekt typu `Node`. Obdobne je v zozname uložený zoznam hrán grafu. Ukladanie odkazov na jednotlivé objekty, a nie len ich identifikátorov, výrazne urýchli prístup k položkám a prechádzanie celej štruktúry pri postupnom vykonávaní algoritmu.

5.2 Vytvorenie kontrolnej siete

Generovanie kontrolnej siete je, ako už bolo spomenuté v kapitole 4.1, jednou z najdôležitejších súčasti používaného algoritmu aj samotnej implementácie. Aby bolo možné využívať knižnicu samostatne, bez nutnosti riadenia užívateľom, boli vytvorené funkcie na automatické vytvorenie siete. Avšak ako uvidíme v nasledujúcich kapitolách, výsledky nemusia byť vždy dostatočné. Preto bol v práci kladený doraz na to, aby bolo možné porovnať ako dosiahnutý výstup, zlúčenie hrán, priamo súvisí práve s týmto krokom algoritmu. Preto je možné celú sieť vytvoriť ručne, a dosiahnuť tak lepší vizuálny výsledok.

Vypočítavanie kontrolnej siete je možné rozdeliť na dve základné oblasti – práca s mriežkou, vytvorenie buniek a regiónov, a následne samotné vytvorenie siete a nájdenie dôležitých bodov, cez ktoré budú vedené hrany finálneho grafu.

5.2.1 Generovanie mriežky

Všetky potrebné metódy pre prvú časť, vytvorenie a zlučovanie buniek, sa nachádzajú v zdrojových súboroch v triede `Grid`. Do konštruktoru sú predávané tri parametre, ktoré priamo ovplyvňujú rozloženie siete. Pre výpočty je nutné mať načítaný a uložený vstupný graf, jeho vrcholy a hrany. Jednotlivé kroky algoritmu a postupné volanie funkcií sú nasledujúce.

Ohraničenie grafu

V grafe nájdeme krajné body – určíme štyri najkrajnejšie súradnice vstupných vrcholov grafu. Dva vypočítané body, ľavý horný a pravý dolný roh a ich súradnice sú uložené ako objekt triedy `Point`. Na základe týchto bodov určíme ohraničujúci obdĺžnik grafu. Aby hranica neprechádzala priamo krajnými vrcholmi grafu, je hodnota vypočítaných rohov zväčšená (resp. zmenšená) o malú hodnotu.

Rozdelenie na bunky

Nasleduje vytvorenie pravidelnej mriežky, veľkosť jednej bunky je predávaná ako parameter do konkrétnej funkcie, a je ju možné nastavovať pomocou implementovaného rozhrania. Túto veľkosť je nutné zadávať ako celé číslo. Ako budeme môcť vidieť v neskorších častiach práce, tento parameter ma nemalý vplyv na finálny výsledok. Pre každú bunku uchovávame súradnice ľavého horného bodu a index – pozíciu bunky v celkovej mriežke.

Výpočet uhla hrany

Pri ďalších krokoch je nutné mať vypočítaný uhol každej vstupnej hrany grafu. Každá hrana, instancia triedy Edge, obsahuje údaje potrebné pre určenie rovnice priamky v smernicovom tvare:

$$y = kx + q,$$

kde:
x, y sú súradnice bodov,
k je smernica; $k = dy/dx = (y_2 - y_1)/(x_1 - x_2)$, a
q je posunutie po ose y.
Smernicu k je možné vypočítať ako tangens uhla, ktorú zviera priamky s kladnou poloosou x, na základe toho dostaneme práve tento potrebný uhol – smer hrany. Pre priamku rovnobežnú s osou y neexistuje smernicový tvar, v tomto prípade nastavíme všetky hodnoty na nulu. V našich výpočtoch tak pracujeme s uhlami priamok v rozsahu 0 – 180 stupňov, kde hodnotu 90 majú hrany rovnobežné s osou x, hodnotu 0 majú priamky na os kolmé a pod. Všetky výpočty sa vykonávajú už pri načítavaní grafu a vytváraní jednotlivých objektov reprezentujúcich hrany grafu.

Priradenie hrán do buniek

Keď máme vytvorenú mriežku buniek, prejdeme všetky hrany zadaného grafu a určíme, ktorými bunkami prechádzajú. Najjednoduchším spôsobom ako vypočítať všetky priradenia hrán do buniek, by bolo pre každú hranu hľadať prieniky s ohraničením každej bunky, čiže prienik so štyrmi úsečkami. Avšak pre vyšší počet vstupných hrán a veľké rozlíšenie mriežky by výpočty mohli byť príliš časovo náročné. Preto bol zvolený iný prístup – na základe

polohy počiatočného bodu hrany určíme index bunky, v ktorej hrana začína. Následne nájdeme najbližší prienik hrany s x–ovou alebo y–ovou hranicou niektorej z buniek, a na základe toho zmeníme a priradíme hrane ďalší index bunky ktorou prechádza. Algoritmus opakujeme v cykle až kým nedôjdeme ku koncovému bodu hrany. Všetky výpočty, zmeny aktuálneho indexu bunky, a inkrementovanie resp. dekrementovanie súradníc čiar mriežky sa líšia podľa uhla aktuálne spracovávanej hrany, a sú rozdelené do štyroch funkcií – rozdiel je pre hranu kolmú a rovnobežnú s osou x, pre hranu rastúcu a klesajúcu. Pri týchto výpočtoch taktiež ukladáme dĺžku úseku hrany, ktorou hrana zasahuje do každej z buniek.

Určenie hlavného smeru

Ďalším krokom je pre každú bunku určiť hlavný výsledný smer hrán spadajúcich do danej oblasti. Podľa článku [17], z ktorého táto práca vychádza, sa má určiť, či sa v bunke nachádza veľký zhluk vektorov príznakov, tj. daných uhlov hrán. Ak nie, bunka bude ignorovaná v ďalších krokoch. V tejto práci bol zvolený nasledujúci princíp určenia finálneho uhla:

- Ak oblasťou neprechádza žiadna hrana, s bunkou sa ďalej nepočíta.
- Ak je táto hrana len jedna, výsledný smer je rovný uhlu tejto hrany.
- V prípade viacerých prechádzajúcich hrán sú z nich vytvárané zhluky na základe hodnôt ich uhlov. Maximálny rozdiel uhlov je zadaný ako vstupný parameter do programu a priamo ovplyvňuje celkové výsledky implementovaného algoritmu. Vo funkcii je nutné uvažovať a spracovať možnosť, že do jedného zhluku majú byť pridelené dve hrany s hodnotami uhlov napríklad 170 a 10 stupňov (pri zadaní maximálneho rozdielu aspoň 20 stupňov). Po rozdelení hrán na skupiny potom vyberieme jednu, a výsledný uhol celej bunky bude priemer z hodnôt práve v tejto skupine. Vybraný zhluk je ten, ktorý obsahuje najviac prvkov. Ak nastane zhodnosť v počte prvkov u viacerých skupín, využijeme uložené dĺžky hrán z predchádzajúceho kroku.Vybereme zhluk, v ktorom celková dĺžka hrán je väčšia.

Počas implementácie tohto kroku algoritmu boli uvažované viaceré možnosti ako vypočítať celkový smer každej bunky, čo je dôležitý krok pri celkovej tvorbe finálneho grafu. Hodnotu by bolo možné určiť jednoduchým priemerom všetkých uhlov hrán, prípadne váhovým priemerom s váhou udávanou ako dĺžka hrany zasahujúca do konkrétnej bunky. Taktiež pri tvorbe zhlukov je možné zvoliť iné prístupy. Bohužiaľ nie je možné určiť, či je nami vybraná možnosť najlepšia a správna. Popisovaná funkcionalita bola navrhnutá a vylepšovaná na základe dostupných grafických ukážok jednotlivých krokov v článku, z ktorého práca vychádza, a taktiež podľa dosiahnutých výsledkov po dokončení aj zvyšných častí algoritmu.

Zlúčenie buniek do regiónov

Potom čo sme každej bunke priradili finálnu hodnotu smerového uhla, môžeme ju využiť k zlúčeniu susedných buniek do väčších celkov. Pomocou zadaného parametru (tretia užívateľom definovaná hodnota) nastavíme maximálny rozdiel uhlov, aký bude medzi bunkami, ktoré sa nachádzajú v spoločnom regióne. V algoritme postupne prechádzame všetky bunky. Ak ešte nemajú pridelený región, čo značí uložený index, vytvoríme nový. Rekurzívne prejdeme všetky susedné bunky a skúmame, či splňujú daný maximálny uhlový rozdiel. Opäť je nutné brať do úvahy možnosť zlúčenia buniek s malým uhlom a uhlom tesne pod rozsah 180 stupňov.

Získanie dvojice bodov pre každý región

Posledným krokom pri práci s mriežkou je vypočítanie potrebných premenných pre každý región. Ako prvý určíme celkový uhol regiónu, čo bude priemer hodnôt uhlov z každej priradenej bunky. Následne vytvoríme novú úsečku, ktorá je kolmá na vypočítaný finálny smer a prechádza cez spracovávaný región. To, či táto hrana prechádza cez ťažisko oblasti, prípadne jeho najširšiu časť, nie je v článku, z ktorého vychádzame, uvedené. V implementácii je vypočítaný stredný bod oblasti, ktorého súradnice sú priemerné hodnoty z najkrajnejších x–ových, resp. y–ových okrajov regiónu. Cez tento bod vedieme spomínanú kolmicu a nájdeme jej prienik s hranicami regiónu, pre ktorý hranu vytvárame. Dvojica bodov pre každý región tvorí základ množiny kontrolných bodov generovanej siete.

5.2.2 Vytvorenie trojuholníkovej siete

Algoritmus zlučovania hrán pokračuje ďalej volaním funkcií z triedy `Mesh`. Ako vstup sú predané body, ktoré sme vypočítali v predchádzajúcich krokoch. Zároveň je nutné určiť posledný parameter, ktorý môže zmeniť celkové rozloženie konečného grafu. Dokončenie vytvárania mriežky prebieha v dvoch hlavných krokoch.

Zlúčenie bodov

Počet vygenerovaných kontrolných bodov je odlišný pre každý graf a taktiež ho ovplyvňujú rôzne nastavenia popisovaných parametrov. Pri ich veľkom počte by bola výsledná sieť zložitá, čo by malo za následok príliš veľa zakrivení hrán finálneho grafu. Preto je vhodné počet bodov zredukovať, na čo sa využíva práve posledný zadaný parameter – maximálna vzdialenosť, na základe ktorej budú body zlúčené do jedného. Pritom však nespájame dvojice bodov, ktoré ležali v spoločnom regióne. Tým uchovávame približnú pôvodnú štruktúru siete, ktorú sme dostali iba pri vytvorení mriežky, a tiež nám to umožní zjednodušiť ďalšie kroky algoritmu. Podľa zdrojového článku je možné následne použiť vzorkovanie na vygenerovanie viacerých bodov, ak sú ďalšie body potrebné. Na využitie tejto možnosti by bolo dobré mať viacero informácií, za akých podmienok a s akými súradnicami je vhodné konkrétne body vytvoriť.

Triangulácia Delaunay

Ďalším krokom algoritmu je spojenie vypočítaných vrcholov mriežky pomocou triangulácie Delaunay. Popis algoritmu sa nachádza v kapitole 4.2. V práci bol použitý nástroj CGAL [7], čo je súbor knižníc ktoré ponúkajú jednoduchú a efektívnu prácu s množstvom geometrických algoritmov. Pomocou funkcií v balíčku `Triangulations` je možné vypočítať rozličné typy triangulácií pre množinu bodov v dvojrozmernom priestore, vrátane triangulácie Delaunay. V implementácii by bolo možné využiť obmedzenú (constrained) trianguláciu. Hlavný rozdiel je v tom, že ako vstup nie sú zadané len body, ale aj niektoré hrany medzi nimi. Tieto hrany je nutné pri spájaní bodov zachovať, na základe čoho môžeme dostať odlišné výsledky. V našom prípade by sme mohli zachovať pôvodné hrany, ktoré sme dostali vytvorením popisovaných kolmých úsečiek na výsledný smer každého regiónu.

Pri práci s knižnicou je nutné previezť náš typ bodov `Point` na typ `Point_2`. Množina týchto bodov je následne použitá ako vstup do metódy objektu `Delaunay_triangulation_2`. Pre nás potrebným výstupom je zoznam strán trojuholníkov, čiže zoznam vytvorených hrán, ktoré sú určené ako dvojica ich krajných bodov.

Najväčšou výhodou použitia CGAL je jednoznačne rýchlosť vykonania algoritmu. Optimalizácia výpočtov v tejto knižnici sa prejavuje, a je pre nás dôležitá, hlavne pri manuálnom vytváraní kontrolnej mriežky, kedy sa spojenie bodov do trojuholníkov prepočítava s každým odobraním či pridaním nového bodu.

5.3 Zlúčenie hrán grafu

Potom, ako sme vytvorili kontrolnú sieť, pokračujeme výpočtom bodov, cez ktoré vedieme hrany grafu a tým prirodzene tvoríme ich zhluky. Výpočty prebiehajú samostatne pre každú hranu siete, ktoré sú v zdrojových kódoch reprezentované objektami typu `MeshEdge`.

Nájdenie priesečníkov

Prvým krokom je nájdenie priesečníkov hrany siete s každou pôvodnou hranou grafu. K tomu opäť využívame rovnice priamok v smernicovom tvare, pomocou ktorých získame priesečné body jednoduchým výpočtom sústavy dvoch rovníc s dvoma neznámymi. Pri výpočtoch je nutné brať do úvahy všetky možnosti smeru hrán, ktoré môžu byť navzájom rovnobežné, prípadne jedna z nich môže byť kolmá na x–ovú os a v tom prípade neexistuje jej smernica.

Určenie hlavného kontrolného bodu

Následne pre každú hranu siete určíme jeden kontrolný bod, ktorý sa nachádza v strede všetkých získaných priesečníkov. Podľa zdrojového článku je možné použiť zhlukovací algoritmus na prípadný výpočet viacerých kontrolných bodov. V tomto prípade opäť nie je jasne dané, kedy by bolo vhodné počítať jediný, a kedy väčší počet bodov. Pri implementácii bolo experimentované s viacerými možnosťami. Jednou z nich bolo práve využiť vytváranie zhlukov bodov, a každej hrane tak priradiť bodov viac. V tom prípade sa však stratí účel ich vytvárania – hrany grafu budú prechádzať rôznymi bodmi, a nenastane tak požadované spájanie ich ciest. Ďalšou možnosťou je vypočítanie iba jedného bodu, ale inými spôsobmi, než je jednoduchý priemer krajných súradníc (napríklad je možné priblíženie hlavného bodu bližšie k väčšiemu zhluku priesečníkov, prípadne zasadenie tohto bodu do stredu najväčšieho zhluku).

Vytvorenie nových ciest

Záverečným krokom je priradiť získané stredné body jednotlivým hranám pôvodného grafu. Hrana bude prechádzať každým bodom, ktorý bol vytvorený na hrane kontrolnej siete, ktorú daná hrana pretína. Všetky tieto body sú uložené v zoznamoch a nakoniec zoradené podľa poradia, v akom bude nová cesta vytváraná. Počiatočný a koncový bod sa pre každú hranu v grafe nemení, následne je nájdený bod s najmenšou vzdialenosťou k predchádzajúcemu bodu, a tak postupne pre celý zoznam.

5.4 Výstup

Poslednou súčasťou implementácie je vykreslenie a export upraveného grafu. Aby bol výsledok vizuálne uspokojivejší, sú nové vypočítané hrany vykreslené pomocou kriviek. Graf je následne možné exportovať do vektorového formátu SVG a previesť do rastrového obrázku PNG.

5.4.1 Bézierove krivky

Pri vytváraní konečného grafu a jeho zlúčených hrán sa využíva interpolačný algoritmus na preloženie bodov krivkou. Nové hrany prechádzajú všetkými bodmi, ktoré sme dostali ako výsledok z predchádzajúcich krokov algoritmu. Krivky definujeme pomocou Bezier spline, čo znamená, že sú zložené z individuálnych Bézierových kriviek spojených v hlavných bodoch.

Bézierove krivky [27] sú jedny z najčastejšie využívaných parametrických kriviek využívaných v počítačovej grafike. Sú definované množinou kontrolných bodov, ich počet určuje rád danej krivky. Bézierova krivka obecne prechádza iba svojim prvým a posledným riadiacim bodom, ostatné body iba ovplyvňujú výsledný tvar. V špeciálnych prípadoch krivka môže prechádzať aj ďalšími riadiacimi bodmi.

Ak sú zadané práve dva kontrolné body, krivka je lineárna, a je to jednoduchá priama čiara medzi dvoma bodmi. Pridaním jedného kontrolného bodu dostaneme kvadratickú krivku. Najpoužívanejší typ sú Bézierove kubické krivky, ktoré sú zadané štyrmi riadiacimi bodmi `P1`, `P2`, `P3` a `P4`. Druhý bod, `P2`, určuje smer, ktorým krivka vychádza z počiatočného bodu `P1`; tretí kontrolný bod zase určuje smer, ktorým sa krivka približuje k svojmu cieľovému bodu.

Pre jazyk C++ existuje viacero knižníc, ktoré sa zaoberajú interpolačnými algoritmami na preloženie bodov krvikou. Dostupných je niekoľko metód [4, 9], ktoré pracujú iba s funkciami v kartézskych súradniciach, v tvare y = f(x), čo však nie je vhodné pre náš algoritmus. Taktiež nie je možné využiť metódy, v ktorých je nutné zoradiť množinu vstupných bodov podľa súradníc x. Práve kvôli vyššie zmieňovaným problémom bola vybratá a použitá metóda prevzatá z [23] . Výhodou algoritmu je, že pracuje s ľubovoľnou množinou vstupných bodov a zachováva ich predom definované poradie, čo je nutnou podmienkou v našej aplikácii. V citovanom zdroji bola vyvinutá jednoduchá aplikácia v jazyku C#, ktorá vykreslí vyhladenú krivku nad množinou zadaných bodov v 2D priestore. K tomu využíva práve popisované kubické Bézierove krivky.

Upravenú funkciu bolo možné upraviť a prepísať do jazyka C++, zdrojové kódy sú dostupné pod open licenciou CPOL [1]. Vstupom do funkcie je zoznam n bodov, výstupom je $(n-1)$ dvojíc riadiacich bodov, ktoré sú v ďalšom kroku použité pre tvorbu bézierových kriviek pre export finálneho grafu do formátu SVG.

5.4.2 Export grafu

Ako už bolo spomínané, vytvorený graf z implementovanej aplikácie je možné exportovať do dvoch rôznych formátov.

Formát SVG

SVG – Scalable Vector Graphics je vektorový formát obrázkov pre dvojrozmernú grafiku založený na jazyku XML. Pomocou rozličných elementov je možné definovať tri základné typy grafických objektov – vektorové tvary, rastrové obrazy a textové objekty. Všetky tvary môžu byť formátované pomocou atribútov alebo štýlov CSS. Definícia elementu na vykreslenie kubickej Bézierovej krivky, a jej krajných bodov, je zobrazená na obrázku 5.3.

Pomocou atribútu `d` sa udávajú súradnice štyroch riadiacich bodov v pevnom poradí; písmenom `M` sa označuje štartovací bod, nasledujú dva kontrolné body za písmenom `C`, a

[1] The Code Project Open License , http://www.codeproject.com/info/cpol10.aspx

```
<path d="M10,10 C15,15 20,15 25,10" fill="transparent"
         stroke-width="1.5" stroke="rgb(0,0,0)" />
<circle cx="10.0" cy="10.0" r="3" fill="rgb(0,0,255)" />
<circle cx="25.0" cy="10.0" r="3" fill="rgb(0,0,255)" />
```

Obrázok 5.3: Formát SVG.

koncový bod krivky. Ďalšími atribútmi je definovaná priehľadnosť, hrúbka a farba čiar. Vrcholy grafu sú vykresľované ako vyplnené kružnice s určitým polomerom.

Na vygenerovanie súboru v programe by bolo možné využiť popisované XML parsery, napríklad vytvoriť súbor pomocou DOM štruktúry. Voľne dostupná knižnica pre jazyk C++, `simple-svg` [8] ponúka jednoduché rozhranie na definovanie množstva elementov - základné geometrické tvary, polygóny, text či súradnicový systém pre grafy. Cez parametre je možné nastaviť vlastnosti jednotlivých vykreslených objektov a následne vygenerovať a uložiť požadovaný SVG súbor. Pre nás potrebný element `path` v tejto knižnici chýbal, avšak doplnením novej triedy a definovaním niekoľkých metód je možné využívať všetky ostatné implementované funkcie aj pre tento prvok.

Formát PNG

Ďalšou súčasťou práce je export výsledného grafu do formátu PNG, ktorá však nie je priamo súčasťou programu vytvoreného v jazyku C++. Táto možnosť by bola realizovateľná napríklad využitím knižnice `libRSVG` [13]. Z dostupnej funkcionality tejto knižnice by sme však využili len minimálnu časť, a spracovanie SVG súboru je podmienené priložením ďalších knižníc.

Jednoduchšou možnosťou je tak využitie programu `ImageMagick` [12] a príkazu `convert`, kde ako parametre sú predávané názov vstupného a výstupného súboru. K implementácii je priložený bash script `svgToPng`, v ktorom je využívaný práve popisovaný program, a po jeho spustení a zadaní parametrov prebehne konverzia medzi zadanými formátmi.

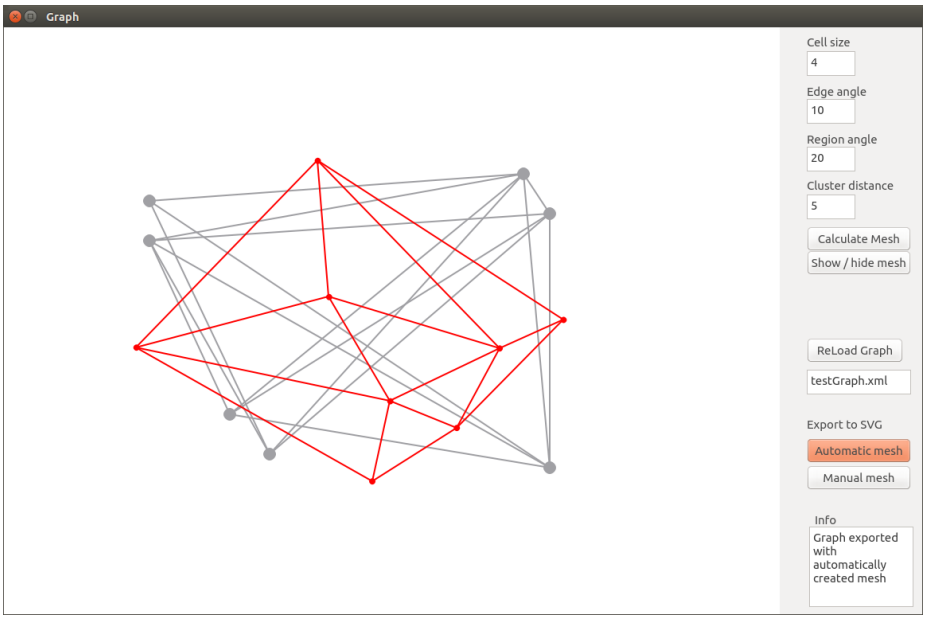
5.5 Užívateľské rozhranie

Aby bola umožnená lepšia prezentácia implementovaného algoritmu, bolo vytvorené jednoduché užívateľské rozhranie (obr. 5.4). Pomocou neho je možné načítať vstupný súbor s grafom a zobraziť ho, zadať parametre do algoritmu, a vytvárať a zobrazovať generované kontrolné siete. K tomu využívame voľné dostupnú knižnicu `Qt` [10], čo jedna z najrozšírenejších multiplatformných knižníc pre vytváranie programov s grafickým užívateľským rozhraním.

Po zobrazení okna je automaticky načítaný vstupný súbor s ukážkovým jednoduchým grafom, ktorý je možné zmeniť pomocou tlačidla `ReLoad Graph`. Automaticky tvorenú sieť môžeme prepočítať s rôznymi parametrami a následne ju vykresliť, prípadne skryť s využitím konkrétneho tlačidla (`Show / hide mesh`). Parametre, ktoré je nutné zadať, sú nasledujúce:

- `Cell size` – celočíselne zadaná veľkosť buniek mriežky.
- `Edge angle` – kladné desatinné číslo, určuje maximálnu veľkosť uhla pri tvorení zhlukov z hrán.

- `Region angle` – maximálna veľkosť uhla pri zlučovaní buniek do regiónov.
- `Cluster distance` – najvačšia vzdialenosť uzlov kontrolnej siete, ktoré budú zlúčené do jedného.



Obrázok 5.4: Užívateľské rozhranie aplikácie.

Jednotlivé parametre a ich presný význam sú detailnejšie popísaný v kapitole implementácia (5.2) . Po každej zmene sa sieť prepočíta až pri potvrdení stlačením tlačidla `Calculate Mesh`. Poslednými komponentami rozhrania sú dve tlačidlá na export grafu do formátu SVG.

Manuálne vytvorenie siete

Ako už bolo spomínané v predchádzajúcich kapitolách, práve kontrolná sieť najviac ovplyvňuje celkový konečný výsledok, a to, ako veľmi sa výsledný graf zjednoduší. Preto je ponúknutá možnosť vytvoriť sieť manuálne a dosiahnuť tak viacero rozličných výsledkov

Na ručné tvorenie siete, zadávanie a odoberanie bodov, je taktiež využívaná knižnica `Qt`. Pridávanie bodov siete je umožnené jednoduchým ľavým kliknutím myši do vytvoreného okna. Po pridaní viacerých bodov je automaticky vypočítavaná triangulácia Delaunay, a body sú tak okamžite spájané do trojuholníkov. Bodmi nie je možné pohybovať a meniť ich pozíciu, avšak pravým kliknutím myši je možné akýkoľvek bod odobrať, a následne ho vykresliť na inom mieste. Potom, čo sú zadané minimálne dva body siete, je možné graf exportovať do súboru, a zobraziť tak finálny graf so zlúčenými hranami s využitím tejto manuálne generovanej siete.

5.6 Jednotkové testy kódu

Súčasťou tejto práce sú taktiež jednotkové testy (Unit Tests), ktoré sa používajú na automatické testovanie a overenie funkčnosti a korektnosti implementácie. V jazyku C++ je

možné k ich tvorbe použiť knižnicu `Boost`, resp. jej časť `Boost Test Library` [1]. Táto knižnica poskytuje kompletné rozhranie pre písanie testovacích programov, je možné rozdeliť testy na viacero jednoduchých prípadov. V nich vytvárame sadu funkcií, ktoré používajú nami implementované metódy. Výsledky testovacieho kódu porovnáme s predpočítanými hodnotami, cieľom testovania je overiť nie len bežné hodnoty a podmienky, ale aj krajné hodnoty a netypické prípady, ktoré môžu počas vykonávania algoritmu nastať.

V našom prípade je vytvorených desať zdrojových súborov s jednotlivými testami. Každý z prípadov pokrýva časť zdrojového kódu, rozdelené sú podľa konkrétnych častí implementovaného algoritmu. Cieľom v tejto práci bolo pokrytie minimálne 80% kódu. K získaniu týchto údajov a štatistík využívame program `geninfo` [3] s využitím príkazu `genhtml`, pomocou ktorého vygenerujeme `html` súbory s údajmi o otestovaných častiach kódu. Výsledky pokrytia sa nachádzajú na obrázku 5.5.

Filename	Line Coverage			Functions	
BezierSpline.cpp		100.0 %	58 / 58	100.0 %	7 / 7
Cell.cpp		100.0 %	65 / 65	92.9 %	13 / 14
DelaunayTriangulation.cpp		100.0 %	16 / 16	100.0 %	3 / 3
Edge.cpp		100.0 %	44 / 44	100.0 %	8 / 8
ExportToSVG.cpp		1.0 %	1 / 98	22.2 %	2 / 9
Graph.cpp		100.0 %	39 / 39	100.0 %	12 / 12
Grid.cpp		94.8 %	218 / 230	100.0 %	16 / 16
Mesh.cpp		100.0 %	49 / 49	100.0 %	10 / 10
MeshEdge.cpp		100.0 %	72 / 72	100.0 %	13 / 13
Node.cpp		100.0 %	2 / 2	100.0 %	3 / 3
ParseXml.cpp		100.0 %	11 / 11	100.0 %	1 / 1
Region.cpp		100.0 %	73 / 73	100.0 %	12 / 12
main.cpp		1.2 %	1 / 80	40.0 %	2 / 5

Obrázok 5.5: Pokrytie zdojových súborov jednotkovými testami.

Ako vidíme, väčšina zdrojových súborov bola testami plne pokrytá. Nízke hodnoty sa vyskytujú pri súbore `main.cpp`, kde sa nachádzajú iba jednotlivé volania otestovaných funkcií na demonštrovanie celého vytvoreného algoritmu. Ďalší súbor, `ExportToSVG.cpp`, obsahuje funkcie na vytvorenie a export finálneho grafu vo formáte SVG. V tejto triede sa využívajú metódy z priloženej knižnice, a tak ich testovanie nebolo pre našu prácu podstatné. Do testovania taktiež nie sú zahrnuté súbory, ktoré využíva `Qt` aplikácia, a v ktorých sa nachádzajú iba metódy pre vytvorenie a prácu s užívateľským rozhraním.

5.7 Zhrnutie

V tejto kapitole boli popísané jednotlivé kroky implementovaného algoritmu. Jednou z hlavných a najrozsiahlejších súčasti je vypočítanie automatickej kontrolnej siete. Ako sme mohli vidieť, výslednú sieť ovplyvňuje viacero parametrov. Nájdenie ich vhodnej kombinácie môže byť pri niektorých zadaných grafoch obtiažne, a ani najlepšie hodnoty nám nemusia zaručiť dosiahnutie dostatočne vylepšeného grafu. Základné kroky algoritmu boli čerpané z už spomínaného zdroja. Bohužiaľ, v tejto publikácií je množstvo postupov popísaných len okrajovo a nie sú dostatočne a presne vysvetlené (konkrétne sú spomenuté v kapitole 5.2). Všetky

funkcie boli počas implementácie jednotlivých krokov navrhované tak, aby čo najlepšie odpovedali vzorovému príkladu v danom článku. Následne, po dokončení celého algoritmu a testovaní ďalších, či už zložitejších, alebo typovo iných grafov, boli spätne niektoré metódy upravované a vylepšované.

Ďalšou nepresnosťou vo vzorovom článku je časť, kde v kapitole pre popis tvorenia automatickej siete je na záver uvedená veta, že tento (automatický) prístup získavania kontrolnej siete je použitý ako predvolený pre zvyšné časti článku. Avšak práve pri popisovaní následných krokov algoritmu na zlučovanie hrán grafu sa vo vzorových obrázkoch využíva sieť iná – použitá pri znázorňovaní tvorby siete manuálne, pomocou ktorej je možné dosiahnuť lepšie finálne výsledky, ako bude ukázané v kapitole s testami (6). Danú sieť nie je kvôli umiestneniu niektorých uzlov možné vygenerovať automatickým spôsobom (resp. by bolo nutné algoritmus upraviť vo viacerých krokoch).

V našej implementácii chýba jedna z častí prezentovaného algoritmu v spomínanom článku, a to konkrétne konečné lokálne vyhladenie grafu. Pomocou tejto funkcionality by bolo možné výsledný graf dodatočne upraviť, a to presunutím niektorých bodov tak, aby hrany nevytvárali príliš kľukaté cesty. Popisovaný algoritmus je zložitejší, a zmysel má hlavne pri zlučovaní hrán pri omnoho väčších grafoch, s množstvom vstupných hrán a uzlov. V celkovej tvorbe tejto práce tak nebol na túto možnosť kladený veľký doraz.

Kapitola 6

Testovanie a dosiahnuté výsledky

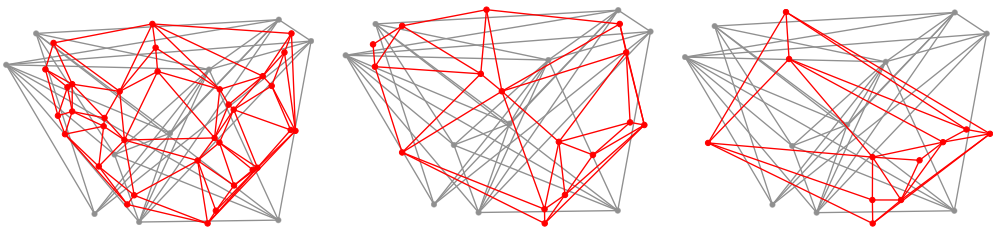
Táto kapitola sa zaoberá testovaním a vyhodnotením výsledkov z vytvorenej aplikácie. V prvej časti sa zameriavame na tvorbu automatickej siete a to, ako je generovanie kontrolnej siete ovplyvnené viacerými parametrami. Následne budú ukázané príklady, ako výsledky vylepšiť manuálnou úpravou bodov siete, a v závere sú prezentované ďalšie dosiahnuté grafy so zlúčenými hranami pre rôzne vstupné dáta. Každý testovací graf použitý v tejto kapitole je priložený k implementácii v popisovanom formáte XML. Pri obrázku sa vždy nachádza názov konkrétneho súboru, a taktiež štvorica hodnôt parametrov, pri ktorej kombinácii sme dostali uvedený výsledok. Tieto čísla postupne značia veľkosť bunky, maximálny rozdiel uhlov hrán, uhlov regiónov, a vzdialenosť pri zlučovaní bodov.

6.1 Vplyv viacerých parametrov pre automatickú tvorbe siete

Ako už bolo spomínané a popísané v kapitole 5.2, tvorba siete je jednou z najväčších častí implementácie. To, aké bude mať daná sieť rozloženie, a teda aký dostaneme finálny výsledok, môžeme ovplyvniť zadaním štyroch parametrov.

Veľkosť bunky

Prvým z nich je veľkosť bunky. Tento údaj je potrebný pri rozdelení grafu na menšie časti, ktoré sa následne zlučujú do regiónov. Čím vyššiu veľkosť bunky zadáme, tým nižšie bude mať mriežka rozlíšenie a buniek bude menej. V tom prípade dostaneme aj menší počet regiónov, čo priamo značí menší počet kontrolných vrcholov a jednoduchšiu sieť. To môžeme vidieť aj na obrázku 6.1, kde sa nachádzajú tri vygenerované výsledky pre jeden vstupný graf. V tomto prípade boli grafy rozdelené na 65 x 43, 22 x 15 a 13 x 9 buniek.

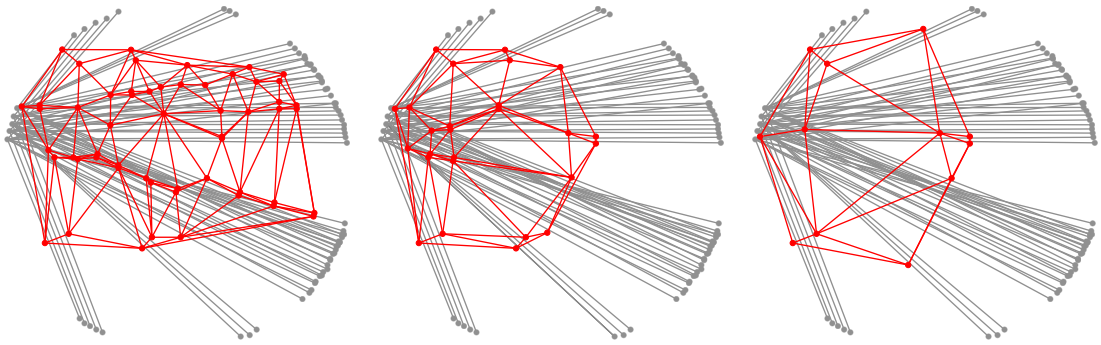


(a) (b) (c)

Obrázok 6.1: Kontrolné siete grafu pri zadaných rôznych veľkostiach jednej bunky, hodnoty postupne 10, 30, 50 (testUsa.xml; i, 20, 20, 50).

Uhol bunky

Ďalším údajom, ktorý je možné pri vytváraní automatickej siete meniť, je maximálna hodnota celkového uhla každej bunky, na základe ktorej sa množina buniek zlúči do väčšieho celku – regiónu. Obdobne ako prvý parameter – veľkosť každej bunky, aj toto číslo priamo ovplyvňuje počet vytvorených regiónov. Ak zadáme malú hodnotu, dostaneme celky s nižším počtom buniek. V tom prípade vytvoríme väčší počet regiónov, kde pre každý z nich je vypočítavaná dvojica bodov, ktoré vo finále tvoria množinu kontrolných bodov siete. A ako už bolo spomínané, ak je kontrolná sieť príliš zložitá, dostaneme väčší počet bodov, cez ktoré musia prechádzať nové hrany grafu, a celkový výsledok je horší. Na obrázku 6.2 sa nachádza trojica automaticky vygenerovaných sietí s rôzne nastavenou hodnotou popisovaného uhla, ostatné hodnoty sa nemenia. V prvej časti môžeme vidieť práve prípad zadania príliš malej hodnoty uhla a jej odpovedajúcu zložitú sieť. S narastajúcou hodnotou tohto parametru sieť zjednodušujeme.



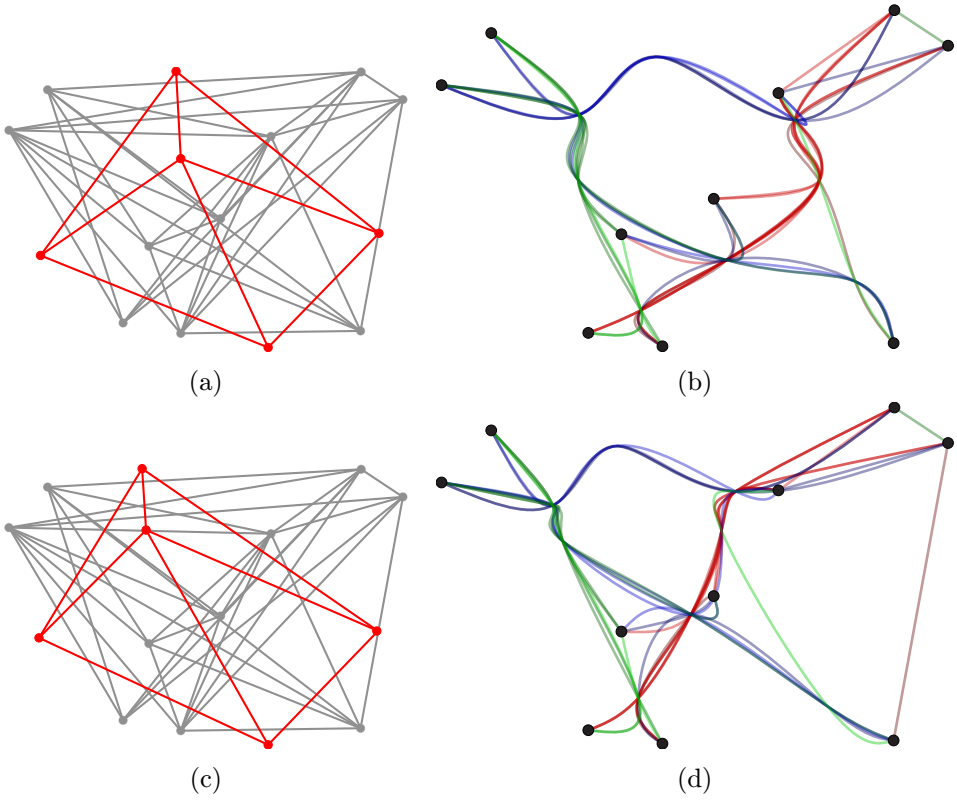
(a) (b) (c)

Obrázok 6.2: Kontrolné siete grafu pri zadaných rozdielnych uhlov pre zlučovanie buniek do regiónov; zľava 2, 5 a 15 stupňov (testTree.xml; 5, 10, i, 20).

Uhol hrán

Graf z prvej ukážky je použitý aj v nasledujúcom príklade, v ktorom vidíme dva rozdielne výstupy pri zmene ďalšieho parametru – EdgeAngle. Táto hodnota udáva maximálny rozdiel uhlov pre vytváraní zhlukov z hrán pri výpočte finálneho uhla každej bunky. V prípade, že je zadané vysoké číslo (k hornej hranici rozsahu 180 stupňov), výsledný uhol je vypočítaný ako priemer zo všetkých hrán prechádzajúcich konkrétnou bunkou. To nie je vhodné napríklad ak máme bunku s prechádzajúcimi dvoma hranami s (takmer) zhodným uhlom, a ďalšou hranou, ktorá do bunky zasahuje len okrajovo a jej smer je úplne odlišný. V tom prípade je lepšie vytvoriť zhluk z prvých dvoch hrán, a pri počítaní výsledného smeru bunky poslednú hranu nebrať vôbec do úvahy.

Na obrázku 6.3 vidíme dva výstupy s minimálnou zmenou tohto parametru – v prvom prípade bola sieť tvorená s hodnotou 19 stupňov, v prípade druhom je hodnota nastavená na 20 stupňov. Aj tak malá zmena spôsobila to, že najpravejší bod siete sa v jednom prípade nachádzal „vo vnútri“ grafu, naľavo od najbližšej hrany. S väčšou hodnotou tak táto hrana nebola zlučovaná s ostatnými, a nebol jej pridelený kontrolný bod, ktorým musí prechádzať. Aj poloha ostatných bodov siete je odlišná, a tak celkové vygenerované grafy sú rozdielne vo viacerých miestach. V ukážke bola použitá jedna z metód na finálne sprehľadnenie grafu a rozlíšenie hrán, a to nastavenie rôznych farieb podľa smerovania (uhla) pôvodných hrán.



Obrázok 6.3: Kontrolné siete grafu pri zadaných rôznych maximálnych uhlov pre tvorenie zhlukov z hrán a získané výstupné grafy; hodnoty 19 resp. 20 stupňov (testUsa.xml; 50, i, 25, 100).

Vzdialenosť pre zlučovanie uzlov siete

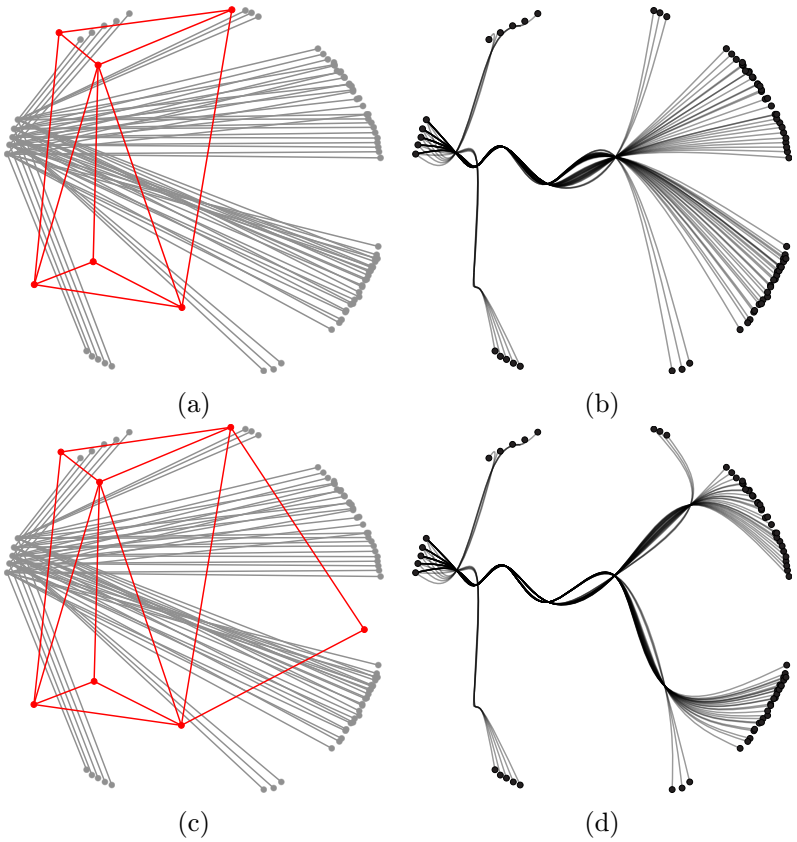
Posledným užívateľom riadeným parametrom je vzdialenosť medzi kontrolnými bodmi siete, na základe ktorej zlúčime viacero bodov do jedného. Tento údaj zjednodušuje mriežku v poslednom kroku jej vytvárania, a vo finálnom grafe je tak menej spoločných bodov, ktorými musia hrany prechádzať. Cesty sú potom menej zložité a kľukatejšie. Avšak v prípade zadania príliš veľkej vzdialenosti sú body zredukované na minimálny počet, a zhluky hrán nie sú tvorené dostatočne.

6.2 Rozdiely pri použití mriežky vytvorenej ručne

V kapitole 5 spomíname možnosť zadať kontrolné body siete ručne, aby bolo možné porovnať dosiahnuté výsledky oproti sieti vygenerovanej automaticky. V niektorých prípadoch stačí malá úprava automatickej mriežky, či už pridanie, odobratie alebo posunutie niektorého z kontrolných bodov, a smerovanie výsledných kriviek v grafe sa zmení a celkový vizuálny výsledok sa zlepší.

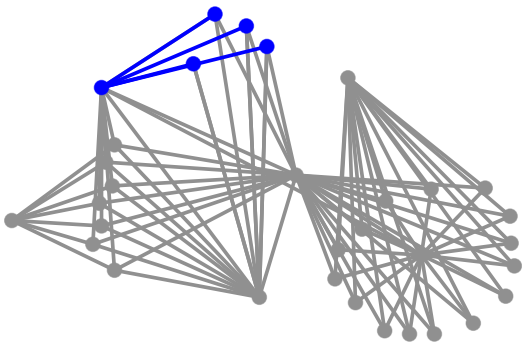
Prvá ukážka znázorňuje jednoduchý graf s jasnou štruktúrou, kde hrany vychádzajú z viacerých miest, ktoré sú umiestnené blízko pri sebe. Na grafy tohto typu sa využívajú viaceré algoritmy, jeden z nich bol popísaný v kapitole 3.1. Na obrázku 6.4 sa nachádza výsledok, ktorý sme získali s využitím nami implementovaného algoritmu. V druhej časti

vidíme rozdielny výstup dosiahnutý s podobnou mriežkou, s pridaním jedného kontrolného bodu navyše. Nastalo tak ešte lepšie zlúčenie hrán a zjednodušenie grafu.



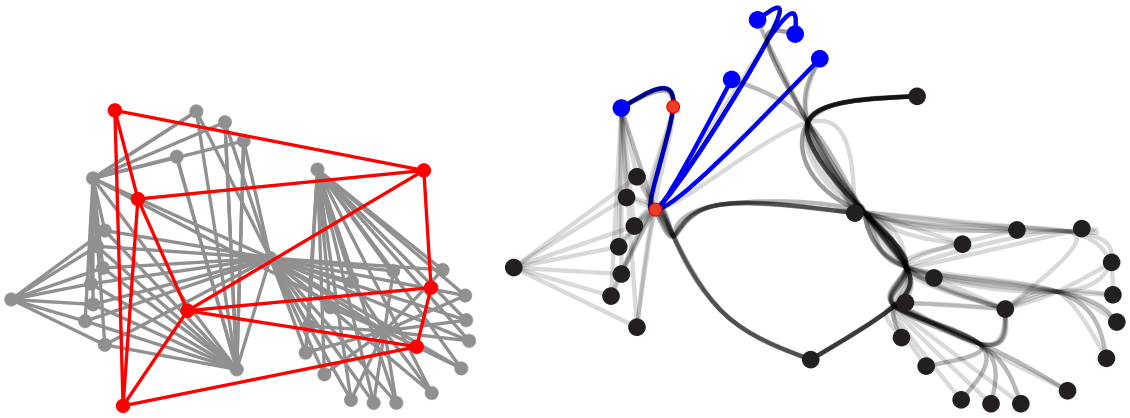
Obrázok 6.4: Rozdielnosť dosiahnutých výsledkov po pridaní jedného bodu navyše do kontrolnej siete (testTree.xml; 30, 20, 20, 50).

Jedným z problémov vytváranej aplikácie s využitím popisovaného algoritmu je finálne tvorenie ciest. Po vykonaní jednotlivých krokov sme pre každú pôvodnú hranu grafu dostali množinu bodov, ktorými dané hrany budeme viesť. Vo viacerých prípadoch nastáva problém, ktorý ukážeme na grafe z obrázku 6.5. Zameriame sa na štvoricu hrán vychádzajúcich z jednoho uzla, ktoré sú zvýraznené modrou farbou.



Obrázok 6.5: Vstupný graf (testWay.xml).

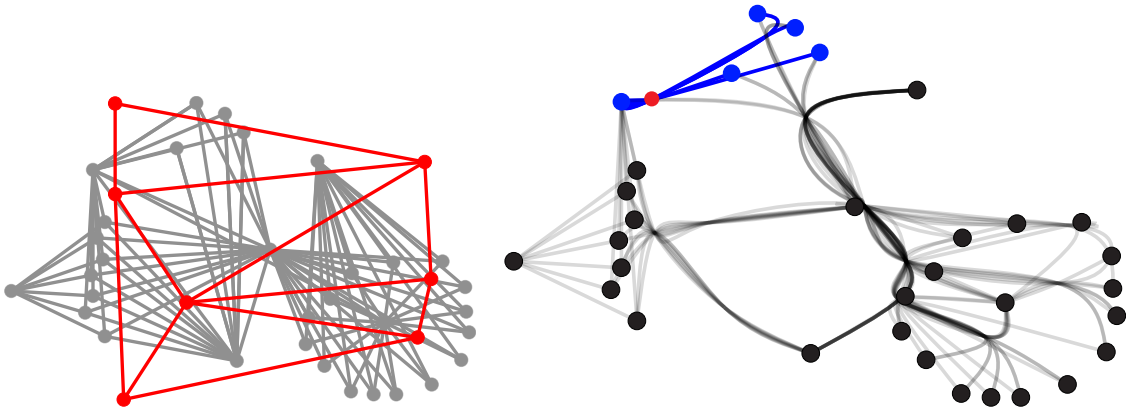
Po vytvorení grafu pomocou automatickej siete dostaneme výsledok zobrazený na obrázku 6.6. Vidíme, že odpovedajúce krivky začínajú a končia v zhodných bodoch ako pôvodné hrany grafu (na modro zvýraznené body), a prechádzajú dvoma riadiacimi bodmi (červená). Poradie riadiacich bodov je určované postupne, prvý bod je vybraný ako najbližší k počiatočnému miestu hrany, ďalšie sú následne usporiadané podľa vzdialenosti k bodu predchádzajúcemu. Tento prístup v niektorých prípadoch spôsobí nesprávne smerovanie hrán, avšak tento stav nie je možné automaticky ošetriť. V ukážkovom grafe by správnejšie smerovanie bolo, ak by hrana viedla najprv nadol, a následne cez druhý kontrolný bod k vrchnému, cieľovému uzlu.



(a) (b)

Obrázok 6.6: Znázornený problém nesprávneho smerovania hrán cez príslušné kontrolné body (testWay.xml; 60, 23, 23, 70).

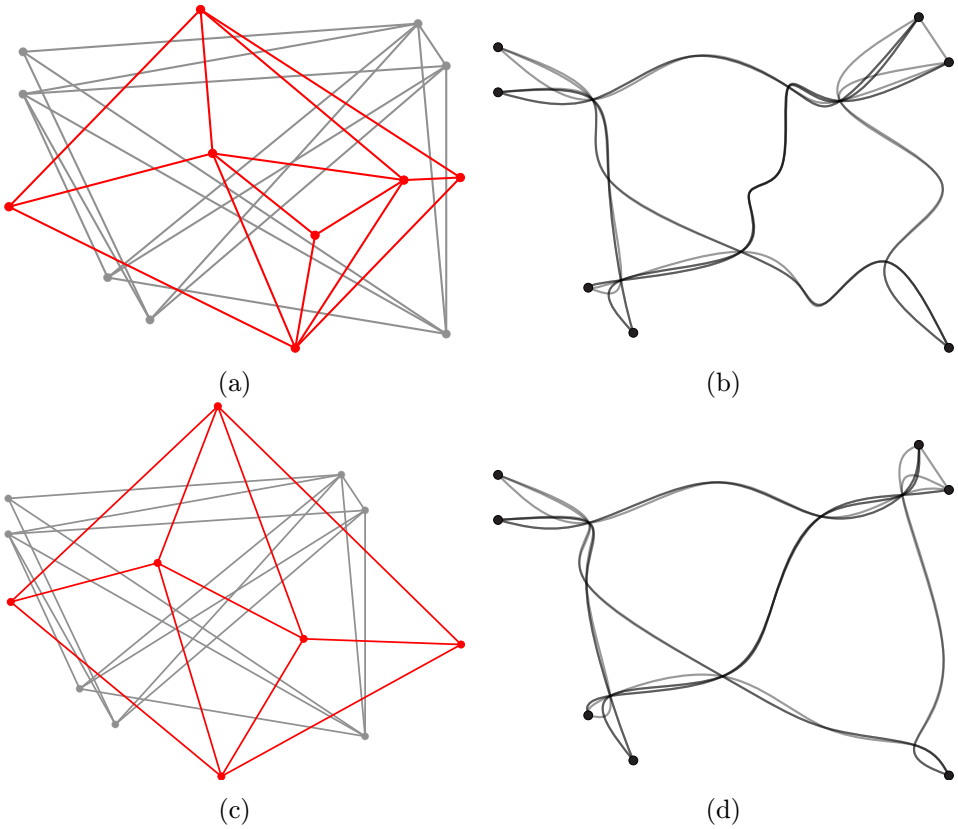
Riešením by mohlo byť úplne odstránenie jedného kontrolného bodu, či prípadne iba jeho presunutie. Taktiež je možné upraviť kontrolnú sieť, napríklad rozdelením najľavejšej hrany siete na dve časti – pridaním jedného kontrolného bodu siete navyše, alebo, ako je uvedené na obrázku 6.7, posunutím jedného z bodov. Smerovanie hrán sa zmenilo a celkový výsledok je lepší.



(a) (b)

Obrázok 6.7: Úprava mriežky pre dosiahnutie lepšieho smerovania hrán. (testWay.xml).

Nasledujúca ukážka zobrazuje jednoduchý graf, ktorý bol používaný ako predloha pri implementácii jednotlivých častí celého algoritmu. Jeden z vizuálne najlepších dosiahnutých výsledkoch pri použití automaticky generovanej siete je zobrazený v prvej časti na obrázku 6.8. Ako vidíme, graf obsahuje viacero kľukatých ciest, čo by bolo možné upraviť zmenou pozícií niektorých kontrolných bodov cez ktoré sú cesty vedené. V druhej časti obrázku sa nachádza výsledný graf pri použití ručne tvorenej siete.

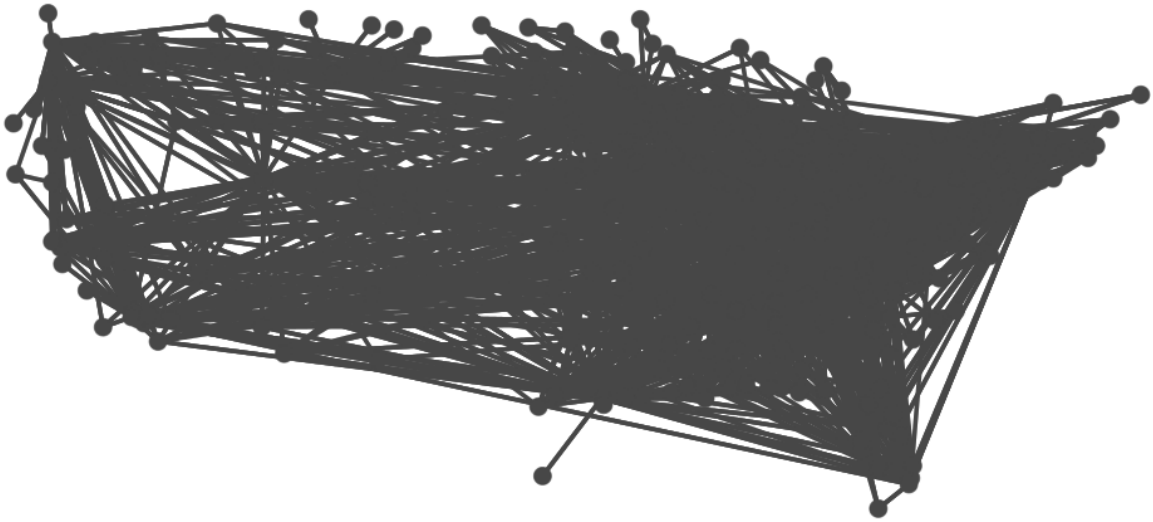


Obrázok 6.8: Testovanie jednoduchého ukážkového grafu. Hore – vytvorená automatická sieť a dosiahnutý výsledok, pod ním zadaná sieť ručne a príslušný zlúčený graf (testGraph.xml; 4, 20, 25, 6).

6.3 Ďalšie ukážky testovacích grafov

Na nasledujúcich obrázkoch môžeme vidieť ďalšie dosiahnuté výsledky na niekoľkých zložitejších vstupných grafoch. Všetky výstupné zobrazenia boli dosiahnuté s využitím automatického prístupu, užívateľsky je nutné zadať iba popisované parametre.

Vstupné dáta z grafu uvedeného na obrázku 6.9 boli prevzaté z internetovej stránky [2], zdrojové kódy ku konkrétnym ukážkam sa nachádzajú na [11]. Pôvodný súbor bol vo formáte XML, avšak bola tu zavedená odlišná štruktúra pre definovanie uzlov. Prvým krokom tak bola najskôr úprava vstupných dát do nami potrebného formátu, k implementácii je priložený finálne spracovaný súbor. Tento graf obsahuje 235 uzlov a 2100 hrán, a znázorňuje trasy leteckých liniek naprieč Amerikou.

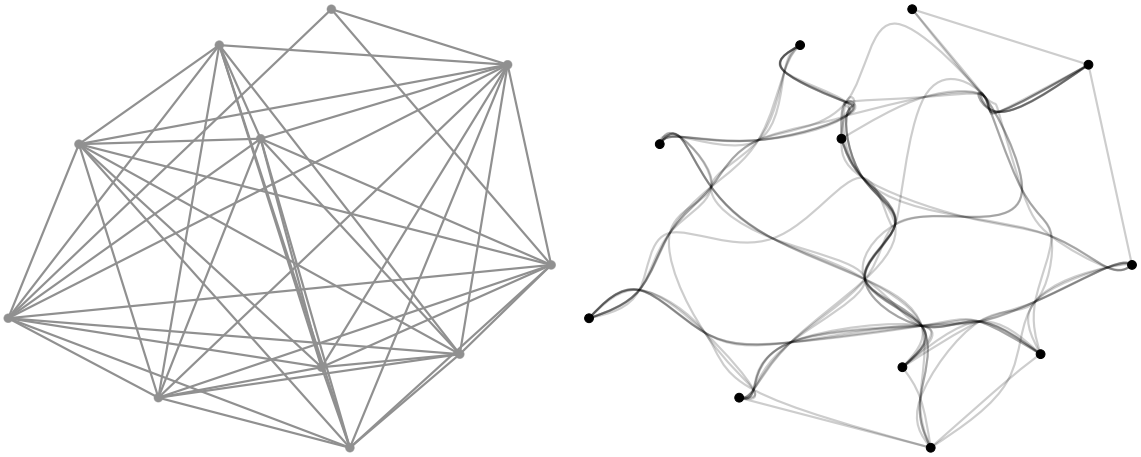


Obrázok 6.9: Vstupný graf znázorňujúci letecké linky v USA (testUsRoutes.xml).

Na obrázku 6.10 vidíme dosiahnutý výsledok po aplikovaní nami vytvoreného algoritmu. V zobrazení nastalo zlepšenie, avšak stále je tu možné dosiahnuť lepšie výsledky. Napríklad aplikovaním vyhladzovacej funkcie, na odstránenie prílišného zakrivenia ciest, alebo s využitím pokročilejších algoritmov a funkcií na zlučovanie jednotlivých hrán.

Obrázok 6.10: Graf so zlúčenými hranami (testUsRoutes.xml; 10, 5, 5, 30).

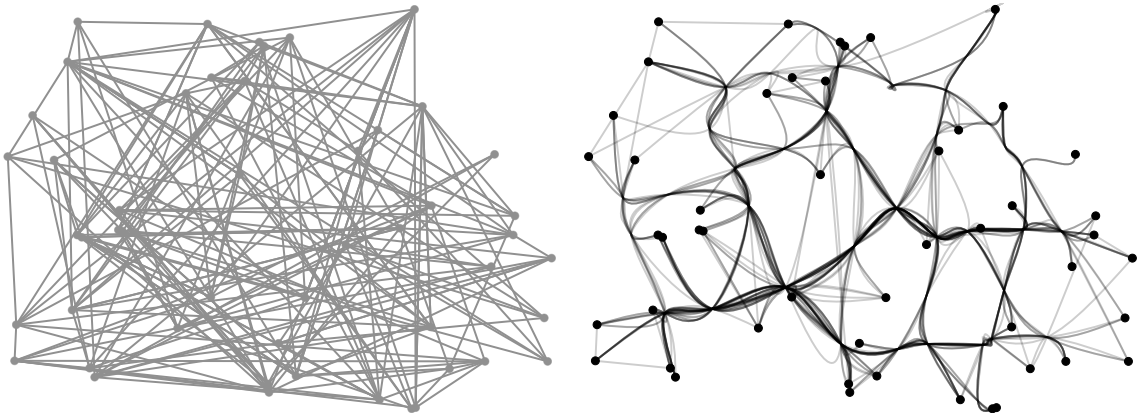
Ďalšou ukážkou je graf s (takmer) rovnakým umiestnením uzlov a hrán ako testovací graf z článku popísanom v kapitole 3.4. Opäť môžeme dosiahnuť niekoľko rôznych výsledkov, v závislosti od vygenerovanej kontrolnej siete. Na obrázku 6.11 sa nachádza originálny graf, a pod ním graf so zlúčenými hranami pre jednu vybranú kombináciu parametrov.



(a) (b)

Obrázok 6.11: Ukážka vstupného grafu (hore) a výsledku po zlúčení hrán (dole) (testDel.xml; 28, 20, 20, 80).

Posledným testovacím grafom 6.12, ktorý budeme prezentovať v tejto práci, je graf s 50 uzlami a 200 hranami medzi nimi. Programovo sme náhodne vygenerovali súradnice jednotlivým vrcholom, a následne, taktiež náhodne, sme pre každú hranu určili identifikátor zdrojového a cieľového uzla.



(a) (b)

Obrázok 6.12: Náhodne vygenerovaný vstupný graf a jeho úprava implementovaným algoritmom (testRandom.xml; 50, 20, 25, 130).

6.4 Rýchlosť vykonania algoritmu

Implementovaný algoritmus sme aplikovali na viaceré vstupné grafy s rozdielnym počtom hrán a uzlov, či grafy s odlišnou štruktúrou. Samotné vykonávanie algoritmu je dostatočne rýchle, v priebehu implementácie na to nebol kladený hlavný doraz a neboli využívané špeciálne optimalizačné techniky. Počas merania sme využívali priemerný počítač [1]. Pre otesto-

[1] CPU: Intel Core i5-2450M @ 2.50GHz, RAM: 4 GB, OS Ubuntu 14.10.

vanie rýchlosti bol algoritmus opakovane spúšťaný na grafoch s väčším počtom hrán, počet uzlov sme zanechali konštantný na čísle 250. Rozmiestnenie uzlov, a ich prepojenie bolo náhodne vygenerované v programe.

Počet hrán grafu	Priemerná rýchlosť
100	0.731
500	1.565
1 000	1.479
5 000	2.407
10 000	2.698

Tabuľka 6.1: Tabuľka priemerných rýchlostí vykonávania algoritmu pre vstupné grafy s počtom uzlov 250 a rôzne veľkým počtom hrán.

V tabuľke 6.1 sú uvedené rýchlosti (priemerná rýchlosť z dvadsiatich meraní) pre niekoľko vstupných grafov. Tieto údaje sme počítali od načítania vstupného XML súboru až po finálne vygenerovanie grafu vo formáte SVG. Ako môžeme vidieť, rýchlosť algoritmu sa zvyšuje s narastajúcim počtom hrán. Nie je to však jediný parameter ktorý výsledné časy ovplyvňuje, dôležitá je aj samotná štruktúra vstupného grafu a rozloženie hrán. Taktiež pre rozličné grafy dostaneme rôzne veľké kontrolné siete s rozličným počtom hrán. To ma vplyv na rýchlosť v jednom z posledných krokov algoritmu, kedy je nutné zistiť všetky priesečníky hrán pôvodného grafu s vygenerovanou sieťou.

Počas testovania sme algoritmus spustili aj na výrazne väčšom grafe, ktorý obsahoval 10 tisíc vrcholov a 100 tisíc hrán. Samotný algoritmus prebehol v poriadku, avšak pri takomto počte nie je možné vidieť takmer žiadne výsledky. Priemerný čas behu programu bol 13.5 sekundy. Pre zaujímavosť, pri tomto grafe je veľkosť vstupného XML súboru 5.1 MB. Po zlúčení hrán mal SVG súbor približne 74 MB a cez 520 tisíc riadkov, takže už len jeho otvorenie a zobrazenie výsledného grafu je problematické.

6.5 Zhrnutie

V tejto kapitole sme prezentovali niekoľko prípadov využitia implementovaného algoritmu. Z ukážok je zrejmé, že pre jednoduchšie grafy, s menším počtom uzlov a hrán, dosiahneme uspokojivé výsledky, a zobrazenie grafov sa skutočne vizuálne vylepší. Pri využití automatického prístupu je jedným z najväčších problémov nájdenie najvhodnejšej kombinácie vstupných parametrov. Lepšie výsledky je možné dostať využitím ručne zadanej siete, ktorej zadanie však pri zložitejších grafoch nemusí byť jednoduché a vyžaduje viac času. Implementovaný algoritmus je možné použiť rozlične veľké grafy, avšak s pribúdajúcim množstvom uzlov a hrán môžu byť dosiahnuté výsledky menej uspokojivé.

Kapitola 7

Záver

Táto práca sa zaoberá rozličnými možnosťami ako upraviť štruktúru grafu a zobraziť ho. Základné metódy sú tie, ktoré vylepšia rozloženia grafu tým, že zmenia pozície vrcholov, čím sa vylepší prehľadnosť a je jednoduchšie zistiť z grafu potrebné informácie o jednotlivých cestách. Následne sú popísané metódy, pomocou ktorých iba odkryjeme lokálne časti grafu a dostaneme tak údaje, ktoré by inak mohli byť skryté. Pokročilejšími algoritmami sú tie, v ktorých nemeníme rozloženie uzlov, ale zhlukujeme hrany do zväzkov, čo môže výrazne znížiť ich kríženie a celkový vizuálny ruch pri vykreslení grafov.

Pokračujeme detailnejším popisom vybranej metódy, geometrickým zhlukovaním hrán grafu. Nachádza sa tu popis jednotlivých krokov, pomocou ktorých zmeníme hrany v hustých grafoch z priamych liniek na krivky. Nové hrany prechádzajú cez spoločné body, čím vytvoríme zhluky kriviek a celkové zobrazenie grafu sa vylepší. Táto metóda má viacero výhod oproti ostatným popisovaným algoritmov – základom je, že nemení rozloženie uzlov, čo je dôležité pre grafy v ktorých majú vrcholy pevne dané umiestnenie. Zároveň je algoritmus vhodný pre akýkoľvek typ grafu.

Dôležitejšou časťou práce je kapitola, v ktorej sú podrobnejšie popísané samotné kroky implementácie algoritmu. Výsledkom tejto práce je súbor funkcií v jazyku C++, pre prezentovanie práce s naimplementovanými metódami sú vytvorené dve aplikácie. Ich vstupom je graf špecifikovaný vo formáte XML, ktorý je prehľadný a zároveň jednoduchý na vytvorenie. Ako sa ukázalo, XML súbor je správna voľba, keďže jeho načítanie a spracovanie v programe je vďaka dostupným knižniciam jednoduché a rýchle.

Do algoritmu je nutné zadať štyri parametre, ktoré priamo ovplyvňujú možné dosiahnuteľné výsledky. Ich detailný popis sa nachádza v príslušných kapitolách. V závere môžeme vidieť ukážky, ako sa výsledné grafy menili v závislosti práve na týchto atribútoch. Počas implementácii a testovania sa ukázalo, že nájdenie príslušnej kombinácie parametrov pre niektoré vstupné grafy má zásadnú rolu k získaniu dostatočne vylepšeného grafu.

Takto dosiahnutý graf je následne uložený vo vektorovom formáte SVG, ktorý je možné prekonvertovať do rastrového obrázku PNG pomocou priloženého scriptu. Samotná konverzia v zdrojových súboroch jazyka C++ by vyžadovala použitie ďalších externých knižníc, preto bolo zvolené takéto riešenie s využitím systémového programu.

Jedným z vytvorených programov je jednoduchá konzolová aplikácia, kde je ako parameter zadaný názov vstupného a výstupného súboru a práve popisované parametre algoritmu, ktoré je možné meniť. Viacero možností však ponúka aplikácia využívajúca Qt toolkit s užívateľským rozhraním. Pomocou nej je možné zobraziť vstupný graf, vykresliť dôležitý medzi krok algoritmu, vygenerovanú kontrolnú sieť, na základe ktorej sú hrany spájané do zhlukov. Zadaním ručnej siete je možné dosiahnuť odlišné výsledky, a porovnať ich s automatickým

prístupom. To, ako sa výsledky líšia pre vytvorené grafy je taktiež možné vidieť v príslušnej kapitole s ukážkami.

Podľa prezentovaných výsledkov vidíme, že sme dosiahli vylepšenie zobrazenia grafov, a vo viacerých prípadoch je možné lepšie zobraziť hlavnú štruktúru grafov či dôležité body. Ďalším rozšírením, ako algoritmus a celkové výsledky ešte vylepšiť, by bolo použitie finálnej metódy na vyhladenie ciest. Predišlo by sa tým nadmernému kľukateniu kriviek a zobrazený graf by bol ešte viac prehľadnejší. Na rôzne grafy by bolo taktiež možné použiť rozličné techniky na zvýraznenie hlavných uzlov a vytvorených ciest.

Literatúra

[1] Boost test introduction [online]. 2007 [cit. 2015-05-20].
URL http://www.boost.org/doc/libs/1_58_0/libs/test/doc/html/intro.html

[2] Data-Driven Documents [online]. 2013 [cit. 2015-05-22].
URL http://d3js.org/

[3] Geninfo – Generate tracefiles from .da files [online]. 2014-05-27 [cit. 2015-05-20].
URL http://ltp.sourceforge.net/coverage/lcov/geninfo.1.php

[4] Cubic Spline interpolation in C++ [online]. 2014 [cit. 2015-05-08].
URL http://kluge.in-chemnitz.de/opensource/spline/

[5] Graphviz - Graph Visualization Software [online]. 2015 [cit. 2015-05-06].
URL http://www.graphviz.org

[6] Xerces-C++ XML Parser [online]. 2015 [cit. 2015-05-06].
URL http://xerces.apache.org/xerces-c

[7] The Computational Geometry Algorithms Library [online]. 2015 [cit. 2015-05-08].
URL http://www.cgal.org/

[8] Simple-svg [online]. 2015 [cit. 2015-05-08].
URL https://code.google.com/p/simple-svg/

[9] Spline interpolation [online]. 2015 [cit. 2015-05-08].
URL http://www.alglib.net/interpolation/spline3.php

[10] Qt | Cross-platform application & UI development framework [online]. 2015 [cit. 2015-05-14].
URL http://www.qt.io/

[11] GitHub: D3.ForceBundle [online]. 2015 [cit. 2015-05-22].
URL https://github.com/upphiminn/d3.ForceBundle

[12] ImageMagick: Convert, Edit, Or Compose Bitmap Images [online]. 2015 [cit. 2015-05-22].
URL http://www.imagemagick.org/script/index.php

[13] LibRSVG [online]. 2015 [cit. 2015-05-22].
URL https://wiki.gnome.org/action/show/Projects/LibRsvg

[14] Bayer, T.: Rovinné triangulace a jejich využití [online].
URL https://web.natur.cuni.cz/~bayertom/Adk/adk5.pdf

[15] Berg, M. d.; Cheong, O.; Kreveld, M. v.; aj.: *Computational Geometry: Algorithms and Applications*. Santa Clara, CA, USA: Springer-Verlag TELOS, třetí vydání, 2008, ISBN 3540779736, 9783540779735.
URL http://doi.acm.org/10.1145/331499.331504

[16] Carpendale, M.; Cowperthwaite, D.; Fracchia, F.: Making distortions comprehensible. In *Visual Languages, 1997. Proceedings. 1997 IEEE Symposium on*, Sep 1997, ISSN 1049-2615, s. 36–45, doi:10.1109/VL.1997.626556.

[17] Cui, W.; Zhou, H.; Qu, H.; aj.: Geometry-Based Edge Clustering for Graph Visualization. *Visualization and Computer Graphics, IEEE Transactions on*, ročník 14, č. 6, Nov 2008: s. 1277–1284, ISSN 1077-2626, doi:10.1109/TVCG.2008.135.

[18] Dickerson, M.; Eppstein, D.; Goodrich, M. T.; aj.: Confluent Drawings: Visualizing Non-planar Diagrams in a Planar Way. *Journal of Graph Algorithms and Applications*, ročník 9, č. 1, 2005: s. 31–52, doi:10.7155/jgaa.00099.

[19] Gansner, E. R.; Koren, Y.: Improved Circular Layouts. In *GRAPH DRAWING*, Springer, 2006, s. 386–398.

[20] Holten, D.: Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *Visualization and Computer Graphics, IEEE Transactions on*, ročník 12, č. 5, Sept 2006: s. 741–748, ISSN 1077-2626, doi:10.1109/TVCG.2006.147.

[21] Jain, A. K.; Murty, M. N.; Flynn, P. J.: Data Clustering: A Review. *ACM Comput. Surv.*, ročník 31, č. 3, Září 1999: s. 264–323, ISSN 0360-0300, doi:10.1145/331499.331504.
URL http://doi.acm.org/10.1145/331499.331504

[22] Kapoulkine, A.: Light-weight, simple and fast XML parser for C++ with XPath support [online]. 2015-04-10 [cit. 2015-05-06].
URL http://pugixml.org

[23] Oleg V. Polikarpotchkin, P. L.: Draw a Smooth Curve through a Set of 2D Points with Bezier Primitives [online]. 2009-03-24 [cit. 2015-05-08].
URL http://www.codeproject.com/Articles/31859/Draw-a-Smooth-Curve-through-a-Set-of-D-Points-wit

[24] Phan, D.; Xiao, L.; Yeh, R.; aj.: Flow map layout. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on*, Oct 2005, s. 219–224, doi:10.1109/INFVIS.2005.1532150.

[25] Qu, H.; Zhou, H.; Wu, Y.: Controllable and Progressive Edge Clustering for Large Networks. In *Graph Drawing*, *Lecture Notes in Computer Science*, ročník 4372, editace M. Kaufmann; D. Wagner, Springer Berlin Heidelberg, 2007, ISBN 978-3-540-70903-9, s. 399–404, doi:10.1007/978-3-540-70904-6˙38.
URL http://dx.doi.org/10.1007/978-3-540-70904-6_38

[26] Sanaulla, M.: Parsing XML using DOM, SAX and StAX Parser in Java [online]. 2013-05-23 [cit. 2015-05-06].
URL http://blog.sanaulla.info/2013/05/23/parsing-xml-using-dom-sax-and-stax-parser-in-java

[27] Tišnovský, P.: OpenGL evaluátory [online]. 2014-05-18 [cit. 2015-05-08].
URL
http://www.root.cz/clanky/opengl-evaluatory-vypocet-bezierovych-krivek

[28] Wong, N.; Carpendale, S.: Interactive poster: Using edge plucking for interactive graph exploration. In *Poster in the IEEE Symposium on Information Visualization*, 2005.

[29] Wong, N.; Carpendale, S.; Greenberg, S.: Edgelens: an interactive method for managing edge congestion in graphs. In *Information Visualization, 2003. INFOVIS 2003. IEEE Symposium on*, Oct 2003, s. 51–58, doi:10.1109/INFVIS.2003.1249008.

Dodatok A

Obsah CD

Priložené CD obsahuje nasledujúce zložky:

- **EdgeClustering** – Zdrojové súbory ku konzolovej aplikácii na zhlukovanie hrán grafu.
- **EdgeClusteringQT** – Zdrojové súbory ku QT aplikácii s užívateľským rozhraním.
- **svgToPng.sh** – Script na prevod svg súboru do obrázku png.
- **DIP-xklimc00.pdf** – Text práce.
- **DIP-xklimc00** – Zdojové súbory textu práce v jazyku LaTeX.