



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA**
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Michal Lehončák

Vizualizace velkých dat

Katedra softvéru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán

Studijní program: Informatika

Studijní obor: Programování a softverové systémy

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Chcel by som poďakovať vedúcemu tejto práce RNDr. Josefovi Pelikánovi za jeho kreatívne vedenie a veľkú ochotu vždy poskytnúť radu. Tiež by som chcel poďakovať mojim najbližším, ktorí ma neprestajne podporovali.

Název práce: Vizualizace velkých dat

Autor: Michal Lehončák

Katedra: Katedra softvéru a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán, Katedra softvéru a výuky informatiky

Abstrakt: V dnešnej dobe sú dáta neoddeliteľnou súčasťou nášho života. Ich objem každým dňom narastá a často nám ich množstvo bráni v pochopení toho, čo pre nás vlastne tieto dáta znamenajú. Cieľom tejto práce je preto vyvinúť aplikáciu slúžiacu k analýze a vizualizácii veľkých dát. Súčasťou práce je aj prieskum štatistických metód využívaných k redukcii objemu aj dimenzie dát a implementácia vybraných algoritmov v tomto obore. Ďalším cieľom je preskúmať možnosti moderných grafických kariet, keďže ich výkonnosť sa každým rokom násobne zvyšuje. Vizualizácia by mala prebiehať za pomoci grafického procesora s dátami zobrazenými vo forme bodov v 3D priestore a užívateľ by mal mať možnosť si tieto dáta interaktívne prehliadať.

Klíčová slova: vizualizácia veľké data PCA SVD redukcia dimenzie

Title: Big data visualization

Author: Michal Lehončák

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Josef Pelikán, Department of Software and Computer Science Education

Abstract: Nowadays, data are an integral part of our lives. Their volume is growing every day, and it often prevents us from understanding what these data means. The object of this thesis is to develop an application for large dataset analysis and visualization. Thesis also explores statistical methods used to reduce volume and dimensionality of data and implements selected algorithms from this field. Another goal is to explore the possibilities of modern graphics cards, as their performance increases every year. The visualization should use a graphics processor with data are shown as points in point-cloud in 3D space and user should be able to browse this data interactively.

Keywords: visualization big data PCA SVD dimension reduction

Obsah

1	Úvod	3
2	Vstupné dáta	5
2.1	Typy vstupu	5
2.2	Organizácia dát	6
2.3	Datové typy	8
3	Metódy analýzy dát	9
3.1	Redukcia dimenzie	9
3.1.1	Analýza hlavných komponent - PCA	10
3.1.2	Analýza hlavných komponent s využitím SVD rozkladu	11
3.1.3	Multidimenzionálne škálovanie - MDS	12
3.1.4	Závery	13
3.2	Zhluková analýza	14
3.2.1	Hierarchické metódy	14
3.2.2	Nehierarchické metódy	15
4	Vizualizácia	18
4.1	Grafický systém	19
4.1.1	Architektúra a výhody grafického procesoru	19
4.1.2	GPGPU	20
4.2	Programovanie programov grafickej karty	21
4.2.1	API	21
4.2.2	Použitie API OpenGL	21
4.2.3	Vykreslovací reťazec	22
4.2.4	Shadery	23
5	Riešenie a výsledky	26
5.1	Od množiny dát k vizualizácii	26
5.2	Výsledky	27
6	Užívateľská príručka	33
6.1	Pred spustením	33
6.2	Spustenie - vizualizačné okno	33
6.3	Okno nastavení - Wizard	34
6.3.1	Načítanie súboru	34
6.3.2	Transformácia dát	35
6.3.3	Vizualizácia dát	36
7	Záver	38
7.1	Zhrnutie	38
7.2	Možné rozšírenia	38
	Zoznam použitej literatúry	40

A	Programátorská dokumentácia	41
A.1	Užívateľské rozhranie	41
A.2	Vstup a reprezentácia dát	42
A.3	Transformácia dát	46
A.4	Vizualizácia	46

1. Úvod

Žijeme v dobe, kedy majú informácie vysokú cenu a ich množstvo okolo nás každým dňom narastá. Úložné miesto sa dnes stáva vždy viac dostupným a lacným, preto je výhodnejšie namiesto trávenia času rozhodovaním, ktoré dáta uložiť a ktoré nie, jednoducho uložiť všetky dáta, ktoré máme. S narastajúcim objemom a dimenzionalitou dát máme však vždy väčší a väčší problém tieto dáta analyzovať a pochopiť.

Príkladom môžu byť veľké softvérové giganty - Facebook, Google, Amazon a ďalšie. Aj keď tieto spoločnosti nezverejňujú svoje datové kapacity, viacero expertov na dáta tieto údaje odhadujú na základe oficiálnych vyjadrení. V začiatkoch Googlu bolo možné celú databázu webu uložiť na desiatich 4GB diskoch, dnes sa odhaduje, že Google ukladá 10-15 exabajtov dát. Podobne Facebook vo svojich začiatkoch pracoval s dátami niekoľkých tisícok užívateľov a ukladal dáta v rádoch gigabajtov. Dnes pri viac ako 2 miliardách používateľov veľkosť úložiska narástla na stovky petabajtov a rastie každým dňom o niekoľko terabajtov. Ďalším zaujímavým číslom je počet spracovaných dát za časovú jednotku, v prípade Facebooku sa odhaduje na stovky terabajtov každú hodinu.

Analýza dát sa ale netýka iba veľkých softvérových spoločností, nástroje na spracovanie dát sú určite potrebné aj v menších spoločnostiach, kde je cieľom analyzovať návyky a preferencie spotrebiteľov.

Súvisiace práce

V dobe stále väčšej popularity vizualizácie existuje mnoho softvérových projektov, ktoré si kladú za cieľ priniesť užívateľovi čo najlepšie prehľadnutie jeho dát.

Najznámejšou platformou v tomto obore je VTK - Visualization ToolKit. Vďaka implementácii v jazyku C++ dosahuje vďaka optimalizáciám jazyka obvykle vyššiu rýchlosť výpočtov, je ale prenositeľný iba na úrovni zdrojového kódu a neponúka žiadne užívateľské rozhranie. Na to sú väčšinou tvorené aplikácie tretích strán, ktoré používajú metódy tejto platformy. Tieto riešenia ale nie sú prispôbené vizualizácii veľkých dát a pri väčšom objeme často zlyhávajú.

Ďalšími aplikáciami sú online vizualizačné nástroje, napríklad Tableau alebo Qlikview. Veľkým negatívom online riešení je prenos lokálnych dát do vzdialených úložísk, čo môže rapídne spomaliť celý proces vizualizácie. Tieto služby často navyše ponúkajú iba prezentačný typ vizualizácie s minimom interaktivity, čo môže byť v niektorých situáciách nedostačujúce.

Ciele práce

Hlavným cieľom tejto práce bolo vytvoriť framework slúžiaci k vizualizácii veľkých dát. Nástroj by mal pracovať plne offline a poskytovať pokročilú interaktivitu užívateľa. Predpokladané vstupné dáta sú v textovom alebo obrazovom formáte, veľkosť dát sa môže pohybovať aj okolo desiatok až stoviek miliónov položiek. Vizualizácia by mala naplno využívať možnosti moderných grafických

kariet. Ďalším cieľom tejto práce bolo preskúmať základné metódy spracovania a analýzy dát a tiež vybrané metódy implementovať.

Diskusia k riešeniu

Framework je vyvinutý v prostredí .NET, konkrétne v jazyku C#. Tento jazyk sme vybrali najmä vďaka jeho základným vlastnostiam - je to silne typovaný objektovo-orientovaný jazyk, ale aj vďaka dobrej funkcionalite vstavaných knižníc, ako aj širokej podpore knižníc tretích strán. Významným pozitívnym kritériom bola aj vstavaná funkcionalita plánovania vláken

Aplikácia využíva okenné užívateľské prostredie systému Windows pomocou knižnice WinForms, ktorá poskytuje všetky potrebné ovládacie prvky a je tiež intuitívna na vývoj a ovládanie.

Pomocou tohto rozhrania si užívateľ môže vybrať zdroj dát, ktorý môže byť textový alebo obrazový, zvoliť si parametre spracovania aj vizualizácie a nakoniec aj interaktívne prehliadať vizualizované dáta. Za parametre spracovania považujeme možnosť aplikovať implementované metódy redukcie dimenzie alebo zhlukovej analýzy na vstupné dáta, parametrami vizualizácie môže byť nastavenie zobrazovaných atribútov alebo nastavenie farieb.

Interaktívna vizualizácia je uskutočnená s pomocou grafickej karty za využitia rozhrania OpenGL. V prostredí .NET je možné používať OpenGL pomocou zaobalovacích knižníc, ako napríklad OpenTK, SharpGL alebo ďalšie, my sme sa rozhodli práve pre OpenTK vďaka otvorenej licencií a jednoduchému používaniu.

Pri implementovaní metód spracovania dát je potrebné vykonávať výpočty nad maticami väčších rozmerov, k čomu je dostupných veľké množstvo knižníc, avšak nie všetky ponúkajú uspokojivý výkon a rozhranie. Po porovnaní viacerých knižníc sme vybrali knižnicu Math.NET.Numerics, ktorá oproti ostatným (Accord, alglib.net) ponúka uspokojivý výkon aj dobre zdokumentované API.

Implementácii sa podrobne venujeme v programátorskej dokumentácii, ktorá je uvedená v prílohe.

Štruktúra práce

V prvej kapitole sa venujeme samotným dátam, ktoré budeme spracovávať. Riešime ich vstup a tiež ich organizáciu v pamäti. V ďalšej časti popíšeme niektoré štatistické metódy používané k analýze dát, porovnáme ich a v diskusii odôvodníme ich výber k implementácii. V tretej kapitole sa pozrieme na možnosti vizualizácie už uložených dát a na manipuláciu s grafickou kartou pomocou vhodného rozhrania. Na záver ponúkneme užívateľský manuál ako aj zhodnotenie výsledkov projektu.

2. Vstupné dáta

2.1 Typy vstupu

V našej práci predpokladáme dva druhy dát - textové a obrazové. Aj keď sú dnes veľké dáta stále častejšie ukladané v neštruktúrovanej podobe, kedy nie je vopred definovaný datový model a každý objekt môže mať rôzny počet aj typ atribútov, my predpokladáme, že vstupom budú štruktúrované dáta, u ktorých je datový model vopred daný.

Textový vstup

Štruktúrované dáta sú v textovej podobe najčastejšie ukladané vo formáte CSV (comma-separated values), kedy je každý objekt umiestnený na samostatnom riadku a hodnoty atribútov sú oddelené vopred známym oddelovačom. Za tento oddeľovač sme pevne určili znak (';') (bodkočiarka). Dôvodom je fakt, že ostatné znaky majú z nášho pohľadu iný sémantický význam. Znak bodky ('.') totiž signalizuje, ide o reálne číslo vo formáte s plávajúcou desatinou čiarkou a znak (',') používame pre oddelenie jednotlivých hodnôt u atribútov, ktoré ich môžu nadobúdať viac, napríklad u výčtových hodnôt.

	A	B	C	D	E
1	id	creator_user_id	state_id	state_client_utc_time	state_server_utc_time
2	1	1	5	2018-03-20 10:44:23.	2018-03-20 10:44:23.
3	2	1	5	2018-04-05 12:28:34.	2018-04-05 12:28:58.
4	1002	1	5	2018-03-20 11:27:33.	2018-03-20 11:27:33.
5	1003	1	5	2018-03-20 11:28:33.	2018-03-20 11:28:33.
6	1004	1	5	2018-03-20 11:28:33.	2018-03-20 11:28:33.
7	1005	1	5	2018-03-20 11:28:33.	2018-03-20 11:28:33.
8	1006	1	5	2018-03-20 11:28:33.	2018-03-20 11:28:33.
9	1007	1	5	2018-03-20 11:28:33.	2018-03-20 11:28:33.

Obr. 2.1: Vzor dobre štruktúrovaného csv súboru

Obrazový vstup

Zaujímavými dátami k analýze sú aj obrazové dáta, u ktorých máme datový model daný implicitne - každý pixel má svoje celočíselné súradnice a 4 farebné zložky farebného modelu RGBA - červenú, zelenú, modrú a alfa kanál. Medzi ďalšie zaujímavé atribúty k analýze a vizualizácii patria aj atribúty pixelu v iných farebných modeloch, ktoré je možno dopočítať z RGBA modelu. My sme pre ukážku použili transformáciu do modelu HSV, ktorý o pixeli ponúkne informácie o odtieni, sýtosti a jase.



*Obr. 2.2: Ukážka vizualizácie obrazových dát:
Na obrázku vľavo vstupný obrázok zobrazený iba pomocou husto umiestnených bodov,
na obrázku vpravo po použití horizontálneho hranového filtra, na zobrazenie hrán je
využitá aj hĺbka.*

K analýze obrazu je možné využiť práve tieto doplnkové atribúty, napríklad hodnota jasovej zložky v oblasti detekcie hrán, odtieň vo vyhodnocovaní farebnej palety obrázku a mnoho ďalších. O možnostiach vizualizácie obrazových dát môže rozhodovať aj sémantika vstupu, najlepším príkladom v tejto oblasti je vizualizácia výškových máp, kedy môžeme na základe bitmapového vstupu vytvoriť trojrozmerný model pevniny.

2.2 Organizácia dát

Existujú 2 základné prístupy k ukladaniu štrukturovaných dát v pamäti - jednou možnosťou je uprednostňovať objektový pohľad, teda že každý záznam je množina atribútov daný mapovaním kľúč-hodnota, kde kľúč je daný menom atribútu a hodnota môže byť ľubovoľného typu. Tento model je zvyčajne jedinou možnosťou v prípade neštruktúrovaných dát, ale zvykne sa používať aj vo svete tých štrukturovaných.

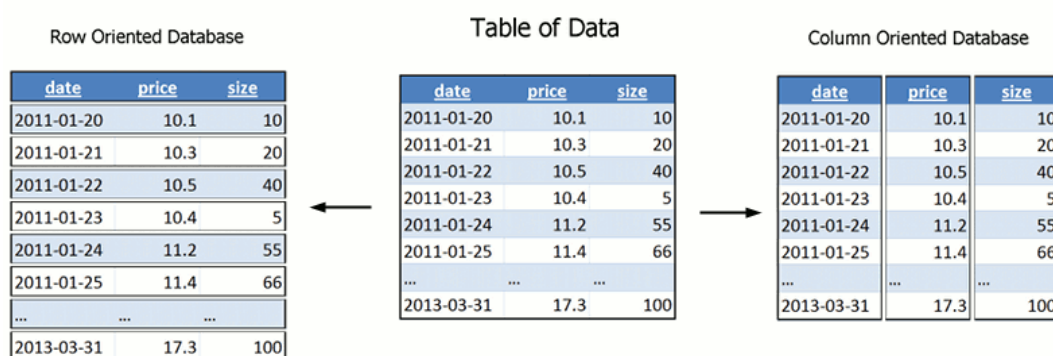
Iným prístupom je, vychádzajúc z predpokladu, že vstupom budú štrukturované dáta, pozeráť na tieto dáta s prihliadaním na homogénnosť dát v rámci jedného atribútu a rovnako ich aj ukladať. Takýto systém organizácie dát sa v prostredí databáz nazýva aj stĺpcovo orientovaná databáza. Záznam potom v takto organizovaných dátach má úlohu balíka, do ktorého sú v prípade potreby zhromaždené všetky atribúty. V našej aplikácii sme použili obe pohľady, každý v inej fáze spracovania dát.

Po načítaní a počas prvej fázy spracovania pri záznamoch hrá kľúčovú úlohu typ ich atribútu, podľa toho sú aj uložené orientovane na stĺpce. V druhej fáze sú preložené atribúty všetkých typov do čísel, kde už typ atribútu nehrá žiadnu úlohu a je možné využiť jednoduchú tabuľku.

Stĺpcovo orientované dáta Vzhľadom na predpoklad štrukturovaných dát, teda že každý záznam má hodnoty u všetkých atribútov a tieto hodnoty sú v rámci

atribútu rovnakého typu, sme náš objekt držiaci dáta v prvej fáze navrhli podľa konceptu stĺpcovo orientovaných databáz, kedy sa na tabuľku dát pozeráme ako na zoznam stĺpcov, ktoré v sebe obsahujú homogénne dáta, na rozdiel od tradičného prístupu, kedy je záznam reprezentovaný zoznamom heterogénnych hodnôt.

Orientácia na stĺpce je v prostredí silne typovaného jazyka C# mimoriadne výhodná, keďže v rámci tohto stĺpca je možné nad daným typom atribútu robiť špecifické pamäťové aj časové optimalizácie, ako aj škálovať alebo normalizovať hodnoty. Stĺpce s číselnými typmi tak obsahujú priamo zoznam obsahujúci hodnoty všetkých záznamov v danom atribúte. Na druhej strane stĺpce obsahujúce výčtové hodnoty, či už jednu na záznam, alebo aj viacero, obsahujú úložisko vo forme mapy, ktoré obsahuje každú výčtovú hodnotu práve raz. Samotný stĺpec potom obsahuje iba zoznam indexov do tohto úložiska.



Obr. 2.3: Schéma porovnania riadkovo a stĺpcovo orientovaných databáz, TimeStored.com

Veľkou výhodou združovania dát v stĺpcoch na rozdiel od riadkov je v prostredí objektových jazykov aj úspora pamäte. Ak by bol totiž každý riadok uložený ako samostatný objekt, okrem svojej hodnoty by v pamäti zaberol ďalších 16 bajtov pamäte, ktoré má v prostredí .NET každý objekt a ktoré sú využívané ako ukazatele na typ objektu a na štruktúru, vďaka ktorej môže objekt vystupovať ako zámok. Tieto bloky pamäte by sme ale neboli schopní nijako využiť a v prípade predpokladaných desiatok až stoviek miliónov záznamov by už tieto bloky pamäte nadobúdali veľký význam.

Ďalším problémom je uloženie heterogénnych dát. Aby mohli byť atribúty objektu uložené v jednom zozname, muselo by byť možné všetky typy atribútov zabaliť do jedného obalového objektu, ktorý by rovnako obsahoval hlavičku veľkosti 16 bajtov.

Inou možnosťou implementácie by bolo zoskupovať atribúty podľa rovnakých typov, čo by nám ale prinieslo pre každý typ atribútu jeden kontajnerový objekt. Toto riešenie je nevýhodné z hľadiska udržiavania kódu a manipulácie s kontajnermi spojenej s hašovaním pri každom prístupe k objektu. V homogénnej štruktúre stĺpca stačí zoznam iba jeden, odpadá nutnosť použiť obalový typ a je tiež možné zoskupovať rovnaké hodnoty, aby neboli ukladané duplicitne.

Riadkovo orientované dáta Riadkovo alebo objektovo orientovaný prístup je využitý vo fáze spracovania dát, kedy je pohľad na záznamy ako na množinu hodnôt atribútov dôležitý. Dôležité je to napríklad pri zhlukovej analýze, kedy

musíme určiť vzdialenosť medzi dvomi objektmi pomocou zadanej metriky, ale aj pri výslednej konverzii na body vizualizácie.

V druhej fáze spracovania sú totiž dáta kódované do číselnej podoby, pri čom u číselných typov je týmto kódovaním priamočiara konverzia do typu `float`, u výčtových typov pre nás už nie je dôležitá presná hodnota, stačí nám že ich vieme odlíšiť čo sa týka rovnosti. V prípade že budú hodnoty rovnaké, bude rovnaké aj ich číselné kódovanie, v opačnom prípade budú hodnoty odlišné. Všetky tieto hodnoty sú uložené do dvojrozsmernej matice reálnych čísel typu `float` implementovanej ako pole objektov, pri čom každý objekt je pole čísel.

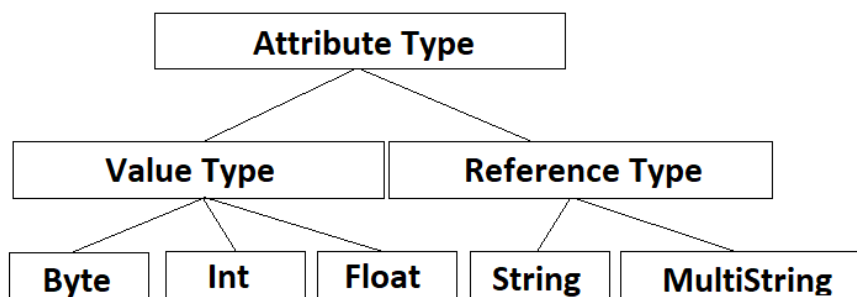
2.3 Datové typy

V našej aplikácii rozlišujeme 5 základných typov dát.

Typy delíme na číselné, teda také, medzi ktorými má zmysel merať vzdialenosť, a výčtové, reprezentované reťazcami, o ktorých vieme povedať iba to, či sa hodnoty rovnajú alebo nie.

Celočíselné typy sú odlišované na menšie bezznamienkové hodnoty typu `byte` s veľkosťou 1B a znamienkové hodnoty typu `int` zaberajúce 4B. Reálne čísla sú ukladané do typu `float` s jednoduchou presnosťou veľkosti 4B. Pomocou týchto 3 typov sme schopní uložiť efektívne akékoľvek reálne číslo.

Výčtové typy sú výlučne ukladané ako reťazce (v .NET typ `string`), dovoľujeme aj viac výčtových hodnôt pre jeden atribút, ak je tento atribút v svojom názve označený ako multiatribút, v takom prípade sa hodnoty rozdelia pomocou znaku `,`. Ak chceme označiť atribút ako viachodnotový, musíme mu na úrovni súboru na koniec mena pripojiť príponu `,-m` alebo môžeme typ atribútu zmeniť manuálne v okne selekcie.



Obr. 2.4: Hierarchické rozdelenie typov v aplikácii

3. Metódy analýzy dát

Úvod

V prípade datasetov s veľkým objemom a veľkým počtom atribútov (veľkou dimenzionalitou) sa obyčajne snažíme tieto vlastnosti zredukovať. V oboch prípadoch redukcie nám môžu pomôcť štatistické metódy, často využívajúce teóriu lineárnej algebry alebo matematickej analýzy.

Veľkým problémom v oblasti vizualizácie veľkých dát je často ich vysoká dimenzionalita, keďže sme schopní intuitívne vizualizovať maximálne 3 dimenzie. Tento problém sa snaží vyriešiť redukcia dimenzie James a kol. (2013), ktorá je populárna nie len v oblasti vizualizácie dát, ale aj v iných procesoch spracovania dát, ako sú napríklad postupy v metódach strojového učenia k zjednodušeniu modelov alebo zrýchleniu výpočtov.

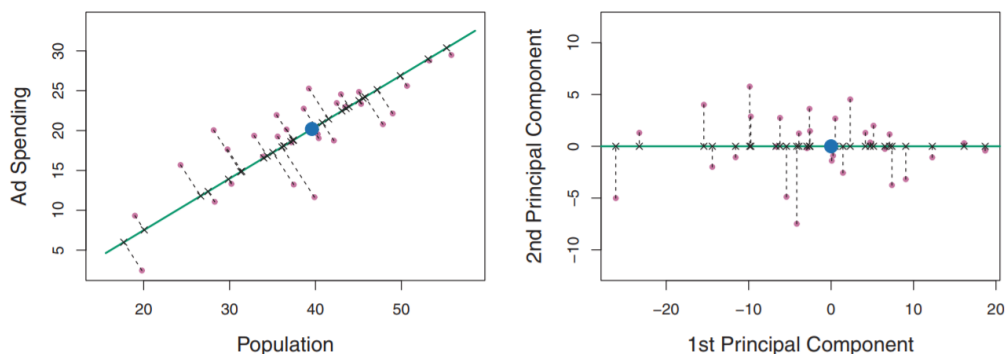
Na zníženie počtu atribútov sa využívajú metódy redukcie dimenzie, pri čom existuje viac druhov metód založených na rôznych princípoch. V našej práci sme sa venovali metódam PCA a PCA-SVD, ktoré sú založené na základných princípoch lineárnej algebry a dáta vnímajú ako maticu reálnych čísel. Tiež sme skúmali aj metódu zvanú multidimenzionálne škálovanie, ktorá na dáta pozerá ako na záznamy, medzi ktorými je možné merať vzdialenosť.

V ďalšej časti sa venujeme redukcii objemu dát. Základnou možnosťou redukcie objemu je filtrovanie podľa podmienky, ktoré vylúči záznamy, ktoré nás pri našej analýze alebo vizualizácii nezaujímajú. Príkladom zo sveta obchodu môže byť vylúčenie zákazníkov, ktorí neuviedli pohlavie alebo je ich vek príliš nízky, aby si kúpili určitý produkt.

Pokročilejšou metódou redukcie objemu môže byť zhluková analýza. Má za cieľ zhromažďovať objekty do skupín (clusterov) podľa podobnosti, pričom podobnosť dát si môžeme definovať svojou vlastnou metrikou. Vo výsledku môžeme nahradiť celý zhluk objektov jedným, ktorý by túto skupinu reprezentoval. Užitočnou funkciou zhlukovej analýzy ale môže byť aj členenie dát podľa podobnosti za účelom demonštrácie práve tejto podobnosti, napríklad ofarbením rovnakou farbou.

3.1 Redukcia dimenzie

Metódy redukcie dimenzie rozdeľujeme do dvoch základných kategórií podľa toho, či vyberajú už existujúce atribúty z tých existujúcich (selekcia atribútov), alebo vytvárajú nové, na ktoré mapujú funkciou prechodu pôvodné hodnoty (extrakcia atribútov).



Obr. 3.1: Ukážka redukcie dimenzie z 2D do 1D - naľavo 2D graf ukazujúci závislosť medzi počtom obyvateľov mesta (*Population*) a výdajmi na reklamu (*ad*). Napravo graf zredukovaný na hlavné komponenty, vodorovne 1. hlavný komponent, zvisle 2. komponent James a kol. (2013)

Selekcia atribútov

Jednou z možností ako dostať požadovaný počet atribútov je výber podmnožiny z už existujúcich. V niektorých prípadoch nám pre skúmanie závislostí medzi atribútmi postačí aj jednoduchá projekcia, teda mapovanie podmnožiny atribútov na 3 rozmery - šírku, výšku a hĺbku, ktoré nám našu množinu dát, zúženú na vybrané atribúty, pomôže zobrazit. Tiež je možné určiť a postupne atribúty vylučovať podľa redundancie.

Extrakcia atribútov

Iným prístupom k redukcii dimenzie je vytvorenie nových atribútov závislých na tých pôvodných. Každá metóda definuje spôsob mapovania, ktoré každej vzorke priradí jej n -dimenzionálnu reprezentáciu, obyčajne sa n rovná 2 alebo 3. Aj keď tieto atribúty zvyčajne neodpovedajú žiadnym z pôvodných atribútov, a teda nemôžeme skúmať konkrétne závislosti, môže nám charakterizovať vlastnosti našej množiny.

3.1.1 Analýza hlavných komponent - PCA

Veľmi populárnou technikou redukcie dimenzie je analýza hlavných komponent (principal component analysis, viď Abdi a Williams (2010)). Táto metóda nájde pre množinu dát dimenzie n lineárny podpriestor dimenzie d taký, že vzorky dát ležia hlavne na tomto podpriestore. Metóda má za cieľ zachovať v tomto podpriestore väčšinu variability. Podpriestor môže byť definovaný pomocou d ortogonálnych vektorov, ktoré zformujú nový súradnicový systém, tieto vektory nazývame hlavné komponenty.

Princíp Nech X je naša vstupná množina dát, daná maticou $t \times n$, kde n je dimenzia dát a t je počet vzoriek. Pre získanie čo najväčšej variancie zvolíme prvý hlavný komponent tak, aby mal maximálny možný rozptyl. Nech teda náš hlavný komponent U_1 je lineárna kombinácia X daná koeficientami $w = [w_1, w_2, \dots, w_n]$,

matematicky

$$U_1 = w^T X$$

a rozptyl je daný

$$\text{var}(U_1) = \text{var}(w^T X) = w^T S w,$$

kde S je výberová kovariančná matica matice X s rozmermi $n \times n$. Z tejto rovnosti vidíme, že rozptyl $\text{var}(U_1)$ môže byť zväčšovaný ľubovoľne manipuláciou s w . My si budeme vektor w vyberať ľubovoľne, ale tak, aby mal stále jednotkovú dĺžku. Chceme teda:

$$\max w^T S w, \quad w^T w = 1.$$

Tento optimalizačný problém riešime pomocou Lagrangeovho multiplikátoru:

$$L(w, \alpha) = w^T S w - \alpha_1 (w^T w - 1),$$

ktorého transformáciou dostávame, že k maximalizácii $\text{var}(U_1)$ sa α musí rovnať najväčšiemu vlastnému číslu matice S a w sa musí rovnať príslušnému vlastnému vektoru S . Z toho môžeme vyvodiť, že prvý hlavný komponent sa rovná normalizovanému vlastnému vektoru kovariančnej matice, ktorý prísluší najväčšiemu vlastnému číslu.

Podobne môžeme dokázať, že d hlavných komponent obdržíme pomocou d dominantných vlastných vektorov.

Algoritmus Algoritmus je možné jednoducho popísať nasledujúcimi krokmi:

1. Získanie výberovej kovariančnej matice - postupne spočítame stredné hodnoty každého atribútu a z vycentrovaných vzoriek (od každej vzorky odpočítame strednú hodnotu) spočítame kovariančnú maticu $n \times n$ celej množiny dát.
2. Spočítame rozklad matice pomocou vlastných čísel a vektorov a vezmeme d dominantných vlastných vektorov - tých, ktorých vlastné čísla sú maximálne. Z týchto vektorov potom zložíme maticu U s rozmermi $d \times t$, čo je matica prechodu z pôvodného priestoru do redukovaného podpriestoru.
3. Každú vzorku x z množiny dát transformujeme získanou maticou prechodu, čím získame t vzoriek s d atribútmi - hlavnými komponentmi.

Výhodami tohto algoritmu je malá časová a priestorová náročnosť, jeho časová náročnosť je $O(tn^2 + n^3)$, avšak algoritmus je prakticky nepoužiteľný v prípadoch, kedy je počet atribútov väčší ako počet vzoriek.

3.1.2 Analýza hlavných komponent s využitím SVD rozkladu

Ďalšou skúmanou metódou redukcie dimenzie je modifikácia PCA, v ktorej sa nepoužije rozklad na vlastné čísla a vektory (Eigenvalue decomposition - EVD), ale jeho zovšeobecnenie - SVD rozklad (viď Dym (2013)). Tento rozklad je možné použiť na maticu akýchkoľvek rozmerov, výstupom sú singulárne čísla a vektory.

Rozklad má podobu:

$$X = U \Sigma V,$$

kde X je naša vstupná množina dát daná maticou $t \times n$, t je počet vzoriek a n je počet atribútov, U je ľavá singulárna matica $n \times n$, Σ je diagonálna matica $n \times n$, ktorá má na diagonále singulárne čísla (odmocniny vlastných čísel) a V je pravá singulárna matica s rozmermi $t \times t$. Na výpočet kovariančnej matice je tu použitá práve matica V , ktorá je závislá na počte vzoriek. Podobne ako v pôvodnej PCA, z kovariančnej matice vyberieme d dominantných vlastných vektorov a použijeme ich na transformáciu pôvodnej množiny dát.

Algoritmus Metódu môžeme popísať nasledovne:

1. Získanie kovariančnej matice - spočítame maticu $t \times t$, ktorá počítá mieru rozptylu medzi vzorkami, nech V je matica d dominantných vlastných vektorov, Σ diagonálna matica odpovedajúca d najväčším singulárnym číslam
2. Výslednú množinu dát získame transformáciou každej vzorky $y = U^T x$, kde U je matica prechodu daná vzťahom

$$U^T = \Sigma^{-1} V^T X^T,$$

y nazveme kódovaním x .

Problémom tejto metódy je závislosť zložitosti na počte vzoriek, keďže potrebujeme spočítať a v jednej chvíli mať uchovanú celú kovariančnú maticu s rozmermi $t \times t$.

3.1.3 Multidimenzionálne škálovanie - MDS

Alternatívnym prístupom ku zmenšovaniu dimenzie je multidimenzionálne škálovanie (multidimensional scaling, viď Cox a Cox (2008)). Táto metóda sa tiež snaží definovať mapovanie priestoru vyššej dimenzie do podpriestoru, avšak nezakladá sa na zachovaní variancie, ale na zachovávaní vzdialenosti medzi vzorkami.

Princíp Nech X je vstupná množina dát daná maticou $t \times n$. Definujme maticu vzdialeností D^X ako štvorcovú maticu $t \times t$, pri čom platí

$$D^X_{i,j} = d_{ij}^{(X)} = d(x_i, x_j),$$

kde d je metrika - funkcia pre každé dve vzorky určujúca ich vzdialenosť z \mathbb{R} , metrika je vždy kladná a symetrická.

Na základe tejto matice vzdialeností MDS hľadá t vzoriek dát v podpriestore dimenzie d , teda maticu Y s rozmermi $t \times d$ tak, aby matice D^X a D^Y boli navzájom podobné. To vlastne znamená nájsť

$$\min_Y \sum_{i=1}^t \sum_{j=1}^t (d_{ij}^{(X)} - d_{ij}^{(Y)})^2.$$

K tomuto výpočtu je možné použiť ľubovoľnú optimalizačnú techniku výpočtu minima.

Výhodou tejto metódy je uchovanie povahy dát pomocou snahy uchovať vzdialenosti medzi vzorkami. Nevýhodou je vysoká časová aj pamäťová zložitosť, keďže už na uchovanie matice vzdialenosti je potrebné kvadratické množstvo pamäti vzhľadom k počtu vzoriek, rovnako kvadratickú zložitosť majú aj algoritmy, ktorými dosiahneme optimalizáciu minima.

3.1.4 Závěry

Cieľom tohto prieskumu štatistických metód bolo nájsť najvhodnejší algoritmus na redukciu dimenzie, ktorý by sme prakticky implementovali v našej aplikácii. V porovnaní skúmaných metód sme ako najlepšiu v tomto ohľade vybrali nemodifikovanú analýzu hlavných komponent PCA vďaka časovej a pamäťovej zložitosti závislej hlavne na počte atribútov, na rozdiel od SVD modifikácie a multidimenzionálneho škálovania, ktoré sú závislé z väčšej časti na počte vzoriek.

3.2 Zhluková analýza

Zhluková analýza (cluster analysis, viď Tan a kol. (2018)) je súbor štatistických metód skúmajúcich podobnosť dát. Jednotlivé metódy definujú mieru podobnosti vzoriek dát a začleňujú ich do skupín podľa miery tejto podobnosti. Výsledkom je systém množín, pre ktoré platí:

$$C = \bigcup_{i=1}^n C_i \quad C_i \cap C_j = \emptyset \leftrightarrow i \neq j$$

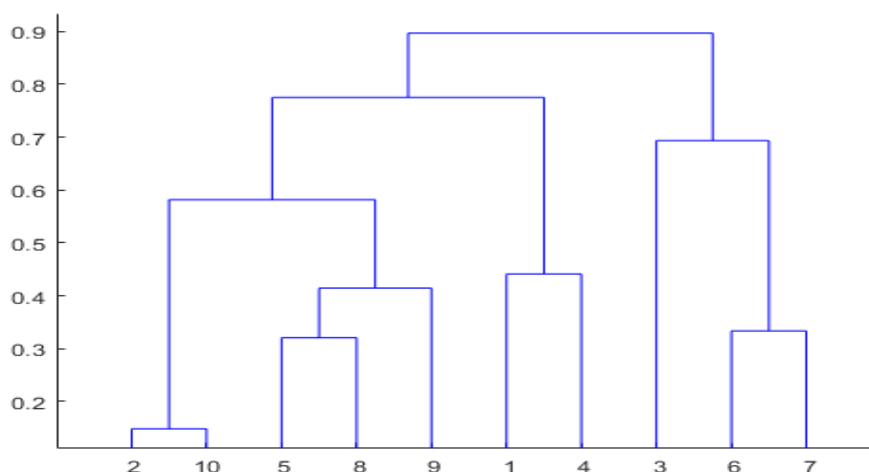
kde C je celá množina vzorkov, C_i je i -tý zhluk a n je počet zhlukov. Počet zhlukov n môže byť vopred daný ako parameter algoritmu alebo je odvodený algoritmom priamo z dát.

Metódy zhlukovej analýzy sa najčastejšie delia do dvoch skupín - hierarchické a nehierarchické.

3.2.1 Hierarchické metódy

Hierarchické metódy sú založené na myšlienke, že vzorky, medzi ktorými je malá vzdialenosť, si budú veľmi podobné. Zhluky sú zoskupené v binárnom strome zvanom dendrogram, kde pre každý uzol platí, že jeho synovia tvoria jeho disjunktný rozklad. Koreň stromu reprezentuje celú množinu vzoriek, naopak listy tvoria samostatné vzorky.

Existujú dva prístupy k hierarchickému zhlukovaniu, a to aglomeratívne a divízné. V aglomeratívnych prístupoch sa buduje stromová hierarchia zdola hore, teda na začiatku algoritmu tvorí každá vzorka zhluk. V každej iterácii algoritmu sa vyberú dva zhluky, ktoré k sebe majú najbližšie, čo sa týka vzdialenosti, a zlúčia sa do jedného zhuku. Naopak v divíznom prístupe postupujeme opačne - začneme s celou množinou dát ako so zhlukom a postupne ho binárne delíme na časti, ktoré sú si viac vzdialené.



Obr. 3.2: Štruktúra vytvorená hierarchickou zhlukovou analýzou zobrazená vo forme dendrogramu, MathWorks

V oboch prípadoch máme v každej iterácii algoritmu korektné rozdelenie na zhluky, ktoré spĺňa, že vzorky, ktoré majú k sebe bližšie, budú v rovnakom zhlukov.

Veľkou výhodou je aj to, že nemusíme vopred špecifikovať počet zhlukov, algoritmus je sám schopný vrátiť množinu zhlukov, ktoré najlepšie charakterizujú vstupné dáta. Tento prístup tiež vedie k deterministickým výsledkom, keďže algoritmus je založený iba na počítaní metriky medzi vzorkami, neobsahuje žiadny prvok náhody.

Zložitosť algoritmu je $O(n^3)$ pri aglomeratívnom prístupe a $O(2^{n-1})$ pri naivnom prístupe k divíznej metóde (aj keď je možné využiť rôzne heuristické postupy k zrýchleniu), čo robí tieto metódy veľmi pomalé už pre stredne veľké množiny dát.

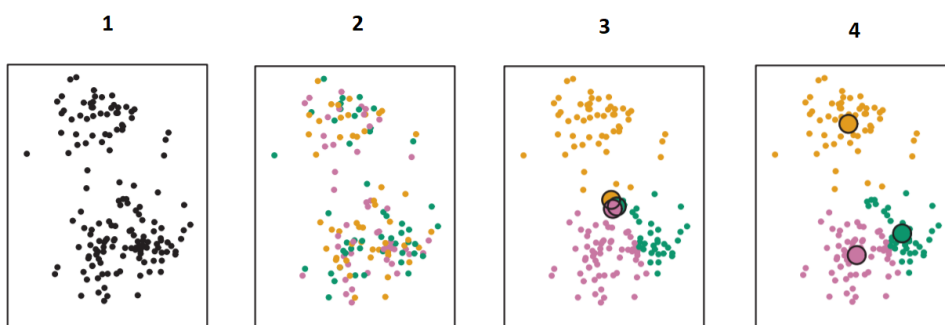
3.2.2 Nehierarchické metódy

Nehierarchické metódy rovnako vytvárajú množinu zhlukov a používajú zadanú metriku ako základ podobnosti vzoriek, ale pri výpočte nepostupujú hierarchicky iteratívne. Tieto metódy môžeme ďalej rozdeliť podľa prístupu k rozdeľovaniu bodov do zhlukov, najčastejšie využívané sú metódy založené na počítaní centroidu (geometrického stredu), počítaní distribučnej funkcie rozmiestnenia vzoriek alebo na základe hustoty vzoriek v priestore.

Nehierarchické metódy zhlukovanie typicky vedú k menej presným výsledkom ako hierarchické, ide skôr o optimalizáciu problému ako o deterministické začleňovanie do zhlukov. Vďaka tomu sú ale použiteľné aj pre väčšiu množinu vzoriek.

K-means metóda

Metóda k-means nepredstavuje konkrétny algoritmus, ale iba abstraktný postup, ako vyrobiť zhluky na základe počítania s centroidmi množín vzoriek. Na vstupe musí mať prirodzené číslo K , ktoré vyjadruje počet žiadaných zhlukov.



Obr. 3.3: Ukážka postupu k-means metódy:

Obrázok 1: Pôvodná množina dát; Obrázok 2: Inicializačný krok - náhodne rozdelenie do zhlukov; Obrázok 3: Spočítanie centroidov a následné rozdelenie vzoriek podľa najbližšieho centroidu; Obrázok 4: Prepočítanie centroidu, možné pokračovať v iterácii alebo skončiť

V inicializačnom kroku sa všetky vzorky rozdelia náhodne do K množín, najčastejšie tak, aby bol počet vzoriek v každej množine rovnaký. Pre každú z týchto množín sa spočíta jeden bod v priestore, ktorý predstavuje centroid - geometrický

stred - vzoriek z tejto množiny, pričom tento bod nemusí nutne patriť do množiny vzoriek. Body sa potom iteratívne prelievajú z jednej množiny do druhej na základe vzdialenosti k centroidom, typicky sa táto vzdialenosť minimalizuje. Hlavná nevýhoda je potreba vstupného parametru, ktorý udáva počet zhlukov, keďže tento limit môže viesť k umelo vytvoreným hraniciam medzi vzorkami.

V porovnaní s hierarchickým zhlukovaním pozorujeme, že hierarchické zhlukovanie je zamerané na optimalizáciu okrajov zhlukov, k-means metódy na optimalizáciu stredov zhlukov. Nevýhodou metódy môže byť aj značný nedeterminizmus, ktorý je spôsobený náhodným začleňovaním do zhlukov v iniciačnom kroku.

Lloydov algoritmus

Typickou a najznámejšou implementáciou k-means metódy je zhlukovanie pomocou Lloydovho algoritmu (viď Morissette a Chartier (2013)). Je založený na myšlienke interakcie vzoriek a množín, vzorka si v každom kroku spočíta, ktorá množina, resp. jej stred je jej najbližšia, a ak sa líši od jej aktuálnej, presunie sa. Každá množina si po iterácii prepočíta centroidy.

Algoritmicky by sme to mohli zapísať nasledovne:

1. Inicializačná fáza - každá vzorka je priradená do náhodnej množiny.
2. Priraďovací krok - pre každú vzorku spočítaj vzdialenosti k stredom všetkých množín a nájdi tú s najmenšou vzdialenosťou, ak je táto nájdená množina iná, ako množina, v ktorej sa vzorka už nachádza, tak presuň vzorku do tejto množiny.
3. Aktualizačný krok - pre každú množinu aktualizuj jej centroid.
4. Ak nenastali koncové podmienky - žiadna vzorka sa nepresunula alebo počet fáz ešte nedosiahol limitu, pokračuje sa znova krokom 2.

Hľadanie najbližšej množiny pre každú vzorku je nezávislé a môže tak prebiehať paralelne, čo je veľkou výhodou tohto algoritmu.

CCPD - Kapacitne obmedzená distribúcia

CCPD (Capacity-constraint point distributions, viď Balzer a kol. (2009)) je algoritmus pôvodne určený na rovnomernú distribúciu bodov v rovine. Podobne ako Lloydov algoritmus sa ale dá využiť aj ako metóda zhlukovej analýzy.

Algoritmus vychádza z myšlienky, že namiesto interakcie vzorky s jednotlivými množinami medzi sebou môžu interagovať priamo množiny a vymieňať si body, ktoré sú pre nich nevýhodné (príliš vzdialené od stredu) za body, ktoré sú naopak nevýhodné pre druhú stranu a výmena by "zvýhodnila obe množiny. Zvýhodnením sa tu myslí zníženie variancie bodov v množine a väčšia koncentrácia okolo stredu.

Algoritmus

1. Inicializačná fáza - každá vzorka je priradená do náhodnej množiny.
2. Prepočítanie centroidov každej množiny.

3. Interakcia dvoch množín - každá si z tej druhej vyberie body, ktoré sú bližšie jej centroidu ako centroidu pôvodnej množiny, po tomto výbere si obe množiny vymenia rovnaký počet bodov. Táto prebieha pre všetky dvojice množín.
4. Ak sa dosiahol požadovaný limit iterácii alebo si žiadne dve množiny nevymenili žiadny bod, algoritmus končí a vracia výsledok. Inak sa opakuje od kroku 2.

Porovnanie

Rovnako ako pri metódach redukcie dimenzie, aj pri prieskume zhlukovej analýzy bolo cieľom vybrať jednu metódu a pokúsiť sa ju zapojiť do transformačného procesu v aplikácii.

Aj keď implementácie k-means metódy predstavujú iba určité optimalizácie problému a je potrebné zvoliť vopred počet zhlukov, čo nemusí byť užívateľovi vopred známe, z pohľadu časovej zložitosti sa ukázali ako výhodnejšie na implementáciu pre použitie na veľkých dátach.

Obe varianty - Lloydov algoritmus aj CCPD algoritmu predstavujú časovo menej náročné postupy zhlukovej analýzy. Každá je však efektívna v inom využití.

Lloydov algoritmus je viac efektívny v prípade, že máme veľké množstvo cieľových zhlukov. Vzhľadom k tomu, že každý si môže nájsť svoju najbližšiu množinu paralelne s ostatnými, algoritmu neprekáža ani príliš veľký počet bodov.

Naproti tomu algoritmus CCPD je závislý na interakcii dvoch množín, ktoré si počas interakcie aktívne modifikujú svoje body. Použitie paralelizmu je trochu obtiažnejšie na naplánovanie, keďže sa jedna množina nemôže vyskytnúť v dvoch interakciách naraz. V každej fáze interaguje každý zhluk s každým, čo vedie ku kvadratickej zložitosti, takže tento algoritmus je značne pomalší pre väčšie množstvo cieľových zhlukov

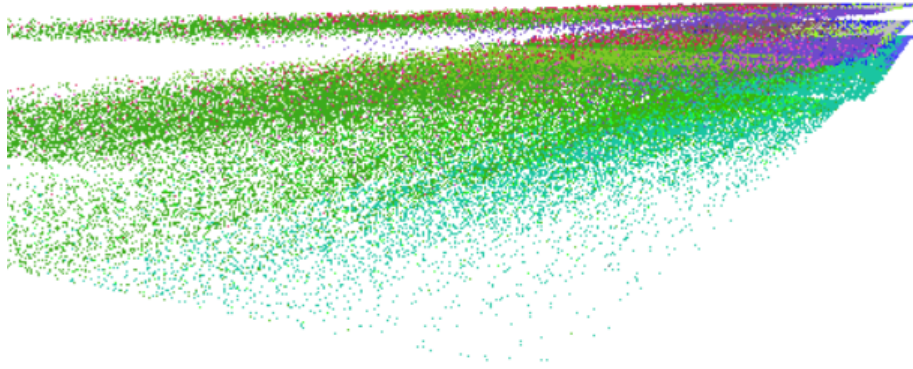
Typicky je Lloydov algoritmus oveľa rýchlejší ako CCPD algoritmus, ale ponúka horšie výsledky. Obrázky porovnávajúce obe metódy na rovnakých dátach uvádzame v kapitole 5.

4. Vizualizácia

Pojem vizualizácia predstavuje súhrn techník, ktoré sú používané k prevodu dát do vizuálnej podoby, obyčajne objektov v n -dimenzionálnom priestore. Nejde v nej však iba o zobrazenie dát v grafickej podobe, ale aj o informáciách skrytých v dátach, v poznávaní štruktúr dát.

V súvislosti s vizualizáciou často hovoríme o dvoch hlavných dôvodoch použitia a to o prezentačnom a exploratívnom. Prezentácia dát zahŕňa aj rozhodovanie akú informáciu vlastne chceme zobrazit, vo väčšine prípadov to závisí na cieľovom publiku. Kvalita spracovania musí byť veľmi vysoká, keďže často ide o finálnu verziu tejto vizualizácie, ktorá bude verejne dostupná ešte dlhý čas.

Na druhú stranu exploratívne vizualizácie slúžia na hľadanie informácii a generovanie nápadov. Obyčajne sa generuje viac snímok, žiadny snímok vizualizácie nie je finálny a každý z nich môže byť buď použitý alebo zahodený.



Obr. 4.1: Ukážka výsledku vizualizácie testovacieho vstupu.

Príkladom prezentačného využitia vizualizácie môže byť graf vytlačený v novinách, musí mať vopred jasné určené, čo chce čitateľovi povedať. Exploratívnu vizualizáciu používame napríklad v našom projekte, kedy pomocou štatistických metód a interakcie s vizualizovaným modelom hľadáme užitočné informácie a závislosti skryté v našej množine dát.

Vizualizácia v trojrozmernom priestore

V našom projekte využívame k vizualizácii kódovanie záznamov do trojdimenzionálneho priestoru, kde je každý záznam reprezentovaný jedným objektom vo forme bodu. Takýto systém vizualizácie sa zvyčajne označuje pojmom „mračno bodov“, anglicky „point cloud“. Základnými vlastnosťami bodu je jeho poloha, daná svetovými súradnicami, a farba, daná modelom RGB.

Po aplikácii redukcie objemu pomocou zhlukovej analýzy, kde nahradíme záznamy pôvodnej databázy stredovými bodmi zhlukov, je možné využiť prvok veľkosti objektu, pričom čím viac pôvodných záznamov patrí do daného zhluku, tým bude zobrazený bod jeho stredu väčší.

4.1 Grafický systém

Pod pojmom grafický systém rozumieme množinu hardwarových komponentov počítača, ktoré sa starajú o grafický výstup. Hlavnou funkciou týchto komponentov je zobrazit' užívateľovi obraz na základe jeho aktivity na počítači, napríklad vykreslovať snímky filmu alebo renderovať hru. Keďže ide o výkonné zariadenie, začalo sa dnes čím viac používať aj na účely iné ako je grafické vykresľovanie. Tieto postupy sú súhrnne označené ako GPGPU a popíšeme ich v samostatnej sekcii.

Grafické systémy môžeme rozdeliť do dvoch hlavných kategórií a to na integrované a dedikované systémy.

Integrovaný systém Ak je grafický systém zabudovaný na iných komponentoch, hovoríme o integrovanom systéme. Systém môže byť integrovaný na matičnej doske alebo na procesore.

Výhodou tohto systému je nižšia energetická náročnosť, nevýhodami je obyčajne niekoľko-násobne horší výkon oproti dedikovaným systémom. Príkladmi takýchto systémov sú grafické procesory zo série Intel HD Graphics, ktoré sú budované k procesorom vydaných spoločnosťou Intel, alebo procesory Adreno dodávané k mobilným zariadeniam v rámci SoC Snapdragon od Qualcommu.

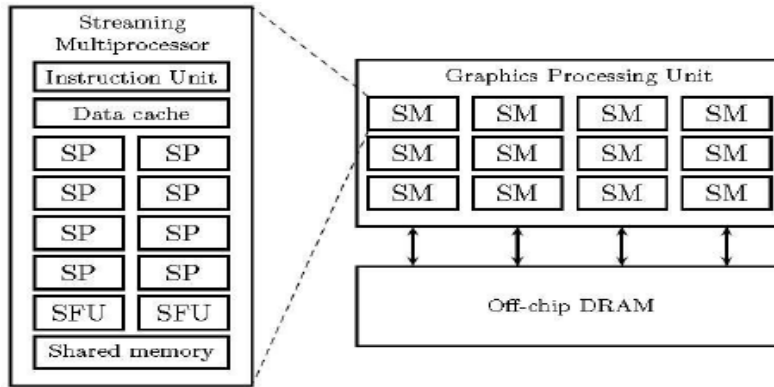
Dedikovaný systém Dedikovaným systémom chápeme systém umiestnený na grafickej karte - komponente systému pripojenému k matičnej doske samostatnou zbernicou PCI Express alebo AGP (Accelerated Graphics Port).

Hlavnou výhodou takéhoto systému je samozrejme omnoho vyšší výkon oproti integrovaným systémom, nevýhodou je vysoká spotreba energie.

4.1.1 Architektúra a výhody grafického procesoru

Grafický procesor (graphics processing unit Weiss (2014)) je mikroprocesor prispôsobený na 2D a 3D renderovanie. Jeho inštrukčná sada je zúžená v porovnaní so všeobecne orientovanými procesormi CPU, pri čom sa zameriava na grafické výpočty. Jeho špecializovanosť na grafické operácie umožňuje prispôbenie architektúry, teda neobsahuje časti, ktoré urýchľujú sekvenčný prúd inštrukcií, ako sú napríklad prediktor skokov alebo komponent prednačítavania pamäte.

Grafické procesory sa vyznačujú masívnym paralelizmom - bežne sa skladajú z viacerých častí zvaných multiprocessory, pričom každý je schopný vykonávať iný prúd inštrukcií. Každý multiprocessor obsahuje ďalej blok výpočetných jednotiek - "streaming"procesorov, dnes zvaných aj jadrá (cores). V multiprocessore sa tiež bežne nachádza zdieľaná (shared) pamäť (zdieľaná výpočetnými jadrami), cache inštrukcií a konštant a malý počet špeciálnych výpočetných jednotiek, určených na počítanie zložitejších matematických výpočtov, napríklad odmocnín a trigonometrických funkcií.



Obr. 4.2: Schéma grafického procesoru. Vpravo vidíme náčrt celého grafického procesoru, vľavo je priblížená jeho časť - multiprocesor.

Jadrá zdieľajú jeden inštrukčný prúd daný inštrukčnou pamäťou ich rodičovského multiprocesoru, ale každý z nich tento program vykonáva nad inými dátami - nad inými vrcholmi alebo fragmentmi. Každé jadro je vlastne multivláknový procesor obsahujúci registrové pole. Implementuje celočíselnú aj reálnu aritmetiku, porovnania, konverzie a tiež špeciálnu jednotaktovú inštrukciu "multiply/add inštrukcia sčítania a násobenia zároveň. Vďaka pipeliningu je možná hardwarová podpora vykonávania viacerých vlákien súčasne, čím odpadá drahá réžia plánovania. Každému vláknu je možné pridelit jeho vlastné registre, od čoho sa odvíja, koľko vlákien môže vykonávať svoje inštrukcie na danom jadre súčasne - je potreba rozdeliť pevný počet registrov medzi maximálny počet vlákien rovnomerne. Výhoda tejto hardwarovej podpory vláknovania je zamedzenie čakania procesoru na pamäť alebo vyhnutie sa pamäťovým či datovým hazardom. V praxi to funguje tak, že ak nejaké vlákno požiada o dáta z pamäte, tak sa v čase odozvy zo strany pamäte vykoná inštrukcia iného vlákna. Tento model sa nazýva SIMT (single instruction, multiple thread).

Ako celok by sme teda grafický procesor mohli začleniť do triedy procesorov MIMD (multiple instruction, multiple data), multiprocesory ale fungujú na princípe SIMD (single instruction, multiple data).

4.1.2 GPGPU

GPGPU je označenie technológie General purpose GPU, čo je súbor metód využívajúcich grafický systém na účely iné ako grafické. Grafický procesor je totiž vďaka svojej schopnosti výkonného paralelného počítania už od začiatku veľmi často využívaný aj na rôzne výpočty, ktoré sú pre všeobecne orientované CPU založené na sériovom počítaní časovo príliš náročné. Väčšinou sa jedná o fyzikálne simulácie, rýchlu fourierovu transformáciu (FFT) alebo o transformáciu obrazu alebo zvuku.

V minulosti bolo jedinou možnosťou vykonávať negrafické operácie jedine za využitia shaderov vo vzkreslovačom reťazci alebo samostatného compute shaderu. Tento spôsob sa ukázal ako nepraktický, keďže musíme mať znalosti programovacieho jazyka shaderov a grafického API.

Moderné prístupy si zakladajú na využití výkonu grafického procesoru z programátorom známych prostredí bez znalostí o shaderoch a API. Dva najznámejšie

frameworky sú CUDA a OpenCL, postavené nad jazykmi C/C++. Programy sa rovnako prekladajú pomocou prekladačov podobných prekladačom pre tieto jazyky, avšak s určitou nadstavbou. CUDA je komerčným produktom spoločnosti NVidia a je teda dostupné iba pre grafické systémy spoločnosti NVidia s podporou CUDA. OpenCL na druhej strane je otvorená platforma pre vývoj aplikácií bežiacich na výpočtovom systéme, ktorý sa skladá z CPU, GPU a ďalších zariadení.

4.2 Programovanie programov grafickej karty

4.2.1 API

Pre programovanie grafickej karty z vysokoúrovňového prostredia je nutné využiť niektorý z frameworkov poskytujúcich API na jej ovládanie, najrozšírenejšími sú dnes DirectX, OpenGL a Vulkan. Najstarším, avšak stále udržiavaným a využívaným, je OpenGL, vytvorený spoločnosťou Silicon Graphics v 90. rokoch a teraz spravovaný konzorciom Khronos Group. DirectX je komplexný balík knižníc vytvorený spoločnosťou Microsoft obsahujúci API pre prácu s grafikou, zvukom, vstupnými zariadeniami a ďalšími, časť určená na prácu s 3D grafikou sa nazýva Direct3D. Vulkan, najnovší z týchto prostredí, považovaný za následníka OpenGL, bol vytvorený konzorciom Khronos v roku 2015. DirectX sa od konkurencie odlišuje proprietárnou licenciou a dostupnosťou iba na systémoch Windows, OpenGL a Khronos sú otvorené prostredia dostupné aj na iných platformách. Khronos je v porovnaní s DirectX a OpenGL viac nízkoúrovňový, ponúka vyšší výkon a vyvažuje využitie CPU a GPU.

Funkcionalita je u všetkých API podobná, my si popíšeme OpenGL, keďže sme ho použili aj v našom projekte.

4.2.2 Použitie API OpenGL

OpenGL (Khronos) je jedným z vyššie predstavených rozhraní pre programovanie grafických kariet. Ponúka veľké množstvo možností prenosu a ukladania dát, ďalej rôzne funkcie na nastavovanie príznakov použitých vo vykresľovacom reťazci a nakoniec ponúka viacero možností vykresľovania objektov aj celých grafických primitív. Tieto časti si následne popíšeme.

Inicializačná fáza

Pri špúšťaní komunikácie s grafickou kartou je potrebné nastaviť globálne parametre vizualizácie. Do tejto inicializačnej fázy tak patrí napríklad preloženie a pripojenie shaderov (transformačných programov pre grafickú kartu, viď 4.2.4), povolenie použitia virtuálnej synchronizácie a nastavenie farby pozadia. Väčšinu týchto parametrov je možné meniť aj v priebehu vizualizácie, pre naše použitie je dostačujúce nastaviť ich raz.

Príprava dát

Základným prvkom prípravy vizualizácie je príprava dát. Typickou reprezentáciou scény v trojrozmernom priestore je množina vrcholov so všetkými atribútmi

na nich naviazanými a množina primitív, teda geometrických objektov, ktoré spájajú dané body. Každý vrchol obsahuje minimálne svoju pozíciu danú reálnym vektorom zloženom zo svetových súradníc, navyše môže obsahovať reálny vektor reprezentujúci farbu v RGB, normálových vektoroch, súradniciach textúry alebo aj o veľkosti vrcholu. Pre definovanie geometrických primitív je možné využiť interné indexy bodov odpovedajúce poradiu uloženia v pamäti, teda na špecifikovanie n -uholníka použijeme n indexov bodov, ktoré tento útvar tvoria.

Na ukladanie dát sú využité objekty typu `Array Object` alebo modernejší `Buffer Object`, ktoré predstavujú neštrukturované bloky dát. Pri vytvorení sa špecifikuje ich budúci spôsob použitia (vrcholy, indexy alebo iné), čo slúži k optimalizáciám, možnosť modifikovať vložené dáta a hlavne veľkosť naalokovanej pamäte v bajtoch. Bajty sú do pamäte ukladané ako neštrukturovaný prúd, ich sémantika sa nedefinuje pri vytváraní ani pri zapisovaní do týchto objektov. Každému z nich je priradený celočíselný index, ktorý je možné použiť na následné špecifikovanie sémantiky pomocou ďalších funkcií.

V našom projekte zobrazujeme dáta výlučne ako vrcholy, použili sme teda iba objekt na uloženie vrcholov, teda `Vertex Buffer Object`. V súvislosti s očakávaním veľkého množstva vrcholových objektov je potrebné optimalizovať dáta, ktoré sa ku každému vrcholu uložia. Ako atribúty vrcholov sme tak zvolili pozíciu vrcholu - 3 reálne čísla typu `float` s veľkosťou 4 bytov - a farbu zakódovanú spolu s veľkosťou vrcholu do 4 bytov, čo je spolu na jeden vrchol 20 bytov.

Hlavným účelom zakódovania farby a veľkosti je pamäťová úspora, keďže najmenším číselným typom v jazykoch shaderov je typ `integer`, jednobytový typ `byte` tu absentuje. Každú z farebných zložiek modelu RGB je ale možné reprezentovať celým číslom medzi 0 a 255, veľkosť vrcholu sa tiež v bežných aplikáciách pohybuje v hodnotách od 0 do 10. Aby sme sa tak vyhli uloženiu 4 jednobytových hodnôt do 4 štvorbytových premenných, tieto 4 byty sme pomocou logických operácií spojili do jedného bezznamienkového čísla typu `uint`.

Vykresľovanie

Samotné vykresľovanie prebieha v dvoch častiach - príprava vstupných dát pre shadere a spustenie shaderov. V rámci prípravy sa presne špecifikuje význam dát uložených v úložiskách typu `Buffer Object` tak, že sa naviažu vstupné parametre prvého shadera v reťazci na jednotlivé byty v úložisku. Pre každý parameter sa špecifikuje jeho offset od začiatku bytov reprezentujúcich jeden vrchol a tiež jeho datový typ, dôležité je tu odlíšiť celé čísla od tých reálnych. V tejto fáze sa rovnako pripravujú aj konštantné hodnoty - `uniforms`, ktoré budú mať rovnakú hodnotu v celom vykresľovacom reťazci. Po nastavení týchto hodnôt je potrebné už len spustiť shadere, vrcholy prejdú vykresľovacím reťazcom a sú vykreslené.

4.2.3 Vykresľovací reťazec

Pod pojmom vykresľovací reťazec (rendering pipeline Shreiner a kol. (2013)) rozumieme model obsahujúci sekvenciu krokov, ktoré je potrebné vykonať, aby sa z 3D modelov stal vykresľovaný obraz v 2D. V minulosti boli tieto kroky pevne dané poskytovateľmi rozhrania (Direct3D, OpenGL), zmena nastala v roku 2001 vydaním DirectX 8.0, ktorá ako prvá priniesla modularitu krokov. Rozširovanie

možnosti programátorov sa stalo populárnym a dnes je už programovateľná väčšina vykreslovacieho reťazca.

Teoretický model sa skladá z modulov, ktoré majú pevne určenú funkciu, avšak môžu mať rôznu implementáciu. Väčšina teoretických modelov sa skladá z týchto fáz:

1. Príprava vrcholov - špecifikovanie 3D scény - vrcholov a zložitejších primitív vo vhodných pamäťových objektoch.
2. Spracovanie vrcholov - aplikovanie užívateľsky zadaných shaderov, ktoré z vstupného vrcholu vyrobí minimálne jeden výstupný vrchol, voliteľne môže byť ale výstupom aj množina primitív. Povinnou fázou je vertex shader - na každý vstupný vrchol sa aplikuje zadaná transformácia. Každému vstupnému vrcholu odpovedá práve jeden výstupný, v tejto fáze nie je možné meniť počet vrcholov. Typickým použitím je projekcia alebo výpočet osvetlenia vrcholu. Voliteľným pokračovaním reťazca je použitie takzvaného "tessellation shaderu", ktorý zjemňuje model rozdelením vrcholov na viacero menších primitív. Po tomto shaderi je možné ešte použiť shader geometrie, ktorým je možné robiť akékoľvek zmeny v geometrii modelu.
3. Post-processing vrcholov - na už užívateľom definované transformované vrcholy sa aplikujú základné geometrické operácie - napríklad odstránenie vrcholov mimo viditeľnosť kamery, zakrytých vrcholov, ale tiež aj normalizácia súradníc a transformácia vzhľadom k výrezu kamery. Tiež prebiehajú prípravy k ďalším fázam.
4. Príprava primitív - zložitejšie grafické primitíva sú rozložené na základné primitíva ako sú body, čiary a trojuholníky a posunuté k ďalšej fázi.
5. Rasterizácia - proces, pri ktorom sa z objektov vo vektorovom formáte (napr. trojuholník) stane množina fragmentov, tiež sa tu rozhoduje o viditeľnosti fragmentov vyrobeného z objektu v rámci výrezu kamery.
6. Spracovanie fragmentov - aplikácia programovateľného fragment shaderu, ktorý pre fragmenty vyprodukované rasterizáciou spočítajú farbu a hĺbku výsledného fragmentu, ktorý sa bude zapisovať do frame bufferu.
7. Spracovanie vzoriek - posledná fáza reťazca, kde sa pomocou viacerých testov a operácii rozhodne, ktoré fragmenty budú zapísané do výsledného bufferu, ktorý budeme vykresľovať. Jedná sa tu napríklad o hĺbkové testy, kombinovanie farieb a ďalšie. Výstupom tejto fázy je frame buffer pripravený na zobrazenie.

4.2.4 Shadery

Shader je program slúžiaci k možnosti modifikovať určité fázy vykreslovacieho reťazca. K jeho vytvoreniu slúžia špecializované programovacie jazyky, medzi ktoré patria GLSL pre OpenGL, HLSL od Microsoftu pre DirectX a Cg od nVidie (dnes už nevyvíjaný), pričom sú si všetky veľmi podobné. Má jeden vstupný bod a množinu vstupných a výstupných premenných. Niektoré druhy shaderov majú

tiež aj povinný výstup daný vstavanými premennými, ktorým musí byť v rámci výpočtu nastavená hodnota.

Reťazec shaderov, ktoré na seba naväzujú svojimi vstupmi a výstupmi, sa nazýva program. Obyčajne má množinu konštant, ktoré sú spoločné pre všetky vyvolania shaderov v rámci jednej dávky, označované ako "constants" v prostredí DirectX, respektíve uniforms" v OpenGL. Každý program pre vykreslovací reťazec musí obsahovať fázy vertex a fragment shaderu, teda transformácie vrcholov a fragmentov, je možné doplniť aj ďalšie druhy shaderov.

Jazyky shaderov sú založené na C++, avšak vzhľadom k tomu, že programy sú vykonávané na grafickom procesore, možnosti sú mierne zredukované, v týchto jazykoch napríklad nie je možné využiť rekurziu. K dispozícii je bežná kontrola toku riadenia, teda cykly, podmienky a switch-case štruktúra, avšak rovnaký kód môže viesť k inému toku pre rozličné vyvolania, čo by znižovalo efektivitu jadier. Za účelom optimalizácie výkonu sa teda odporúča vyhýbať sa zložitým zmenám toku.

Jazyk stavia na vstavaných skalárnych typoch (bool, int, uint, float, double), z nich je možné vytvoriť n-tice - vektory, matice a polia. Práca s vektormi je obzvlášť výhodná vďaka využitiu SIMD inštrukcií. Jazyk tiež umožňuje definovať vlastné štruktúry obsahujúce položky ľubovoľných ostatných typov. Špeciálnu funkciu má vstavaný typ "sampler vzorkovač textúr, kde každý objekt tohto typu odpovedá jednej textúrovacej jednotke. Nad základnými typmi je možné použiť základné aritmetické a logické operácie (abs, pow, sqrt), ale aj zložitejšie geometrické (cross, normalize), goniometrické (sin, cos) a pomocné funkcie často používané pri grafických transformáciách (step, mix).

Druhy shaderov

Podľa funkcionality môžeme shadere rozdeliť na viac druhov, pri čom každý má svoje vstupy, výstupy a obmedzenia.

Vertex shader Vertex shader patrí medzi základné shadere, vo väčšine prípadov povinné. Využíva sa k transformácii vrcholov vzhľadom k projekcii, ale aj k ďalším zmenám atribútov vrcholu, napríklad jeho veľkosti, farby a osvetlenia, s ktorými je možné pracovať v neskorších fázach reťazca. Má teda jeden povinný vstup a jeden povinný výstup, ktorými sú práve súradnice spracovávaného vrcholu.

Fragment/Pixel shader Technicky nepovinným, ale veľmi často používaným, je fragment shader (na DirectX pixel shader). Je spustený na každý fragment vzniknutý vo fáze rasterizácie, hlavným vstupom je pozícia fragmentu v priestore, výstupom je obyčajne farba, prípadne nastavenie hĺbkového alebo iných bufferov, ktoré sú použité pri poslednej fáze - spracovaní vzoriek. Ak nepotrebujeme počítať farbu fragmentu, ale iba hĺbku, čo sa využíva napríklad pri výpočte tieňov, je možné tento krok úplne vynechať, bez použitia fragment shaderu sa totiž hĺbka dopočíta systémovo.

Ďalšie typy Existuje viacero ďalších druhov shaderov, ktoré sú nepovinné a v jednoduchých grafických aplikáciách často nenachádzajú využitie. Do tejto skupiny patrí geometry shader, slúžiaci na zmeny v topológii a geometrii scény alebo skupina tessellation shaderov, ktoré sa starajú o zmenenie topológie. Osobitnú kategóriu tvorí compute shader, ktorý nijako nezapadá do vykresľovacieho reťazca a je možné ho použiť aj na iné, ako na grafické účely, v minulosti plnil funkciu GPGPU.

5. Riešenie a výsledky

V prvej časti tejto kapitoly si predstavíme tok dát v aplikácii a vylepšenia, ktorými sme prispôbovali aplikáciu využitiu pre veľké dáta. V druhej časti si na názorných ukážkach predstavíme praktické využitie preskúmanej teórie.

5.1 Od množiny dát k vizualizácii

Na začiatku každej vizualizácie je výber množiny dát, ktorú chceme prehliadať. Na vstupe môžu byť teoreticky ľubovoľné štrukturované dáta, z ktorých je možné vytvoriť n -dimenzionálne mračno bodov, každá dimenzia tvorí jeden atribút objektu. Vyžaduje sa teda, aby mal každý vstupný objekt definované hodnoty všetkých atribútov. Implementovanými vstupnými metódami sú textový vstup vo forme parseru CSV súborov a obrazový vstup pre bežné obrazové formáty.

Dáta sú po načítaní uložené v stĺpcoch (o dôvodoch využitia stĺpcovo orientovaného pohľadu na dáta sme hovorili v sekcii 2.2). Takto aplikácia efektívnejšie využíva dostupnú pamäť a aj pre väčšie množiny dát nedochádza k jej nadmernému preplneniu.

Po fáze načítania je možné vykonávať nad dátami transformácie pomocou algoritmov zhlukovej analýzy. Zhlukovú analýzu je možné použiť dvomi odlišnými spôsobmi. Prvým je nahradenie objektov v zhluku pomocou geometrického streda zhluku, čo vedie k redukcii objemu dát a zrýchleniu grafickej karty pri vizualizácii. Druhým využitím je prosté označenie bodov indexom zhluku, do ktorého patria. Na základe tohto indexu je možné vo fáze ofarbovania spoločne označiť vrcholy, ktoré sú si podľa výsledkov algoritmu podobné. Zhlukovú analýzu nie je možné použiť v prípade, že dataset obsahuje atribút nadobúdajúci viacero hodnôt, keďže v tomto prípade nie je možné počítat „vzdialenosť“ dvoch množín hodnôt alebo určiť priemer pre potreby spočítania centroidu.

Následne je zo stĺpcovej reprezentácie dát vyrobená tabuľková, ktorá neberie do úvahy hodnoty reťazcov, iba ich odlišnosť medzi jednotlivými bodmi (prejavuje sa odlišným číslovaním).

Konverzia dát do tabuľkovej formy je výhodná pre algoritmy lineárnej algebry nahliadajúce na dáta ako na maticu čísel. Aplikácia umožňuje využiť implementovaný algoritmus slúžiaci na redukcii dimenzie - analýzu hlavných komponent - PCA. Tento algoritmus zredukuje dáta z ľubovoľne vysokého počtu dimenzií do počtu, ktorý je výhodný pre účely vizualizácie, typicky sa dimenzia redukuje na 3 alebo 2 rozmery.

Poslednými krokmi je výber mapovania pôvodného mračna bodov, ktoré je typicky n -dimenzionálne, do 6-rozmerného mračna bodov, ktorého 3 rozmery predstavujú polohu bodu v súradnicovom systéme vizualizácie a ďalšie 3 rozmery udávajú ofarbenie. Toto mračno bodov je teda už možné vizualizovať.

Pre obrazové dáta je navyše možnosť pomocou filtrov detekcie hrán spočítat práve jasovú zložku po aplikácii filtra, čo umožňuje skúmať hrany trojrozmerné. Aplikácia ponúka detekciu hrán pomocou horizontálneho, vertikálneho a obojsmerného filtra.

5.2 Výsledky

Výsledné obrázky našich vizualizácií si ukážeme na našich testovacích vstupoch, ktoré sú pribalené k aplikácii.

Text Textové vstupy reprezentujú 3 súbory typu csv, konkrétne súbor `users` predstavujúci databázu užívateľov internetového portálu, súbor `surnames` predstavujúci zoznam priezvisok obyvateľov USA zoradený podľa výskytu a súbor `evals` uchováajúci množinu hodnotení produktov užívateľmi.

Súbor `users` je klasickým databázovým súborom uchovávajúcim informácie o užívateľoch. Jeho datový model tvorí identifikačné číslo užívateľa, pohlavie, čas založenia účtu, čas poslednej aktivity a napríklad aj počet komentárov a počet priateľov, či priemerné hodnotenie produktov daného internetového portálu.

	A	B	C	D	E	F	G	H	I	J	K	L
1	id	sex	since	last	acq	evals	comments	points	friends	updates	evalAve	evalVar
2	1	1	372779	407867	409143	14106	0	3	0	1	4.841	0.862
3	2	1	342083	418835	418843	19998	380	56	96	15	3.125	1.034
4	3	1	338795	418835	418876	26328	0	26	0	15	2.994	0.886
5	4	0	385619	418091	418768	27862	0	10	13	15	2.752	1.413
6	5	1	375395	419099	419111	34185	0	2	0	15	3.033	0.603
7	6	1	283883	419507	419528	18746	1918	1368	0	16	3.119	0.942
8	7	1	368243	416507	418547	23483	0	9	0	13	3.441	0.957
9	8	1	322019	419555	419668	25801	1	9	0	16	2.614	1.299

Súbor `surnames` je tvorený štatistickými dátami spracovanými v priebehu demografického prieskumu zameriavajúce sa na početnosť priezvisk v USA. Datový model tvorí samotné priezvisko, jeho početnosť, podiel na 100,000 obyvateľov, ale aj podiel medzi jednotlivými rasami vyskytujúcimi sa v USA.

	A	B	C	D	E	F	G	H	I	J	K	L
1	name	rank	count	prop100k	cum_prop	pctwhite	pctblack	pctapi	pctaian	pct2prace	pcthispani	most
2	SMITH	1	2376206	880.85	880.85	73.35	22.22	0.40	0.85	23012	20455	white
3	JOHNSON	2	1857160	688.44	1569.30	61.55	33.80	0.42	0.91	29952	18264	white
4	WILLIAMS	3	1534042	568.66	2137.96	48.52	46.72	0.37	0.78	43102	21916	white
5	BROWN	4	1380145	511.62	2649.58	60.71	34.54	0.41	0.83	31413	23377	white
6	JONES	5	1362755	505.17	3154.75	57.69	37.73	0.35	0.94	31048	16072	white
7	MILLER	6	1127803	418.07	3572.82	85.81	15250	0.42	0.63	11324	15707	white
8	DAVIS	7	1072335	397.51	3970.33	64.73	30.77	0.40	0.79	26665	21186	white
9	GARCIA	8	858289	318.17	4288.50	42887	0.49	15707	0.58	0.51	90.81	hispanic

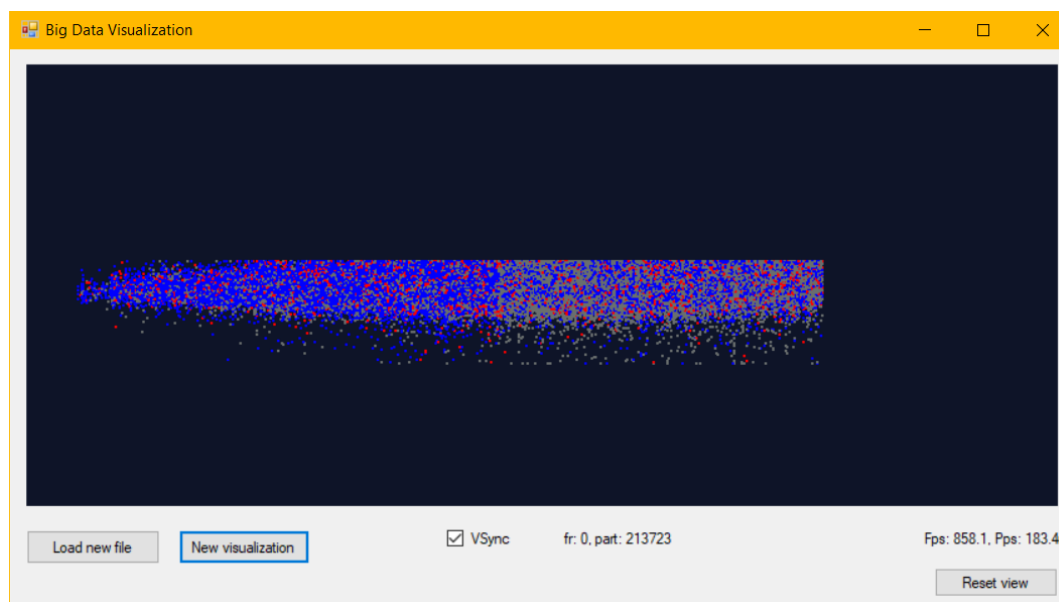
Obr. 5.1: Pri každom priezvisku je uvedený podiel jednotlivých rasových etník, ktoré dané priezvisko nosia, zľava to sú po poradí biele etnikum bez hispánskeho obyvateľstva, čierne etnikum, etnikum ázijsko-pacifické, etnikum pôvodných obyvateľov, príslušníci 2 rás a hispánske etnikum

Predchádzajúce súbory reprezentujú dáta s väčším počtom dimenzií. Súbor `evals` na druhú stranu obsahuje iba 3 rozmery - id užívateľa, id produktu a celočíselné hodnotenie na škále od 0 do 5. Tento súbor však oproti predchádzajúcim dvom obsahuje omnoho viac záznamov, počet sa pohybuje okolo 22 miliónov.

Ďalšie súbory k vizualizácii nechávame na užívateľa. Existuje ale viacero portálov, ktoré sami obsahujú alebo ponúkajú odkazy sa vhodné csv súbory, uvedieme napríklad <https://www.kdnuggets.com/datasets> alebo <http://statweb.stanford.edu/~sabatti/data.html>.

Základnou technikou bez použitia ďalších štatistických metód je jednoduchá projekcia na atribúty, medzi ktorými chceme skúmať závislosť.

Prvou ukážkou je výber sledovaných atribútov zo súboru `users`. Sledovať je tu možné viacero závislostí, my sme sa rozhodli ukázať vo vizualizácii nasledovné:



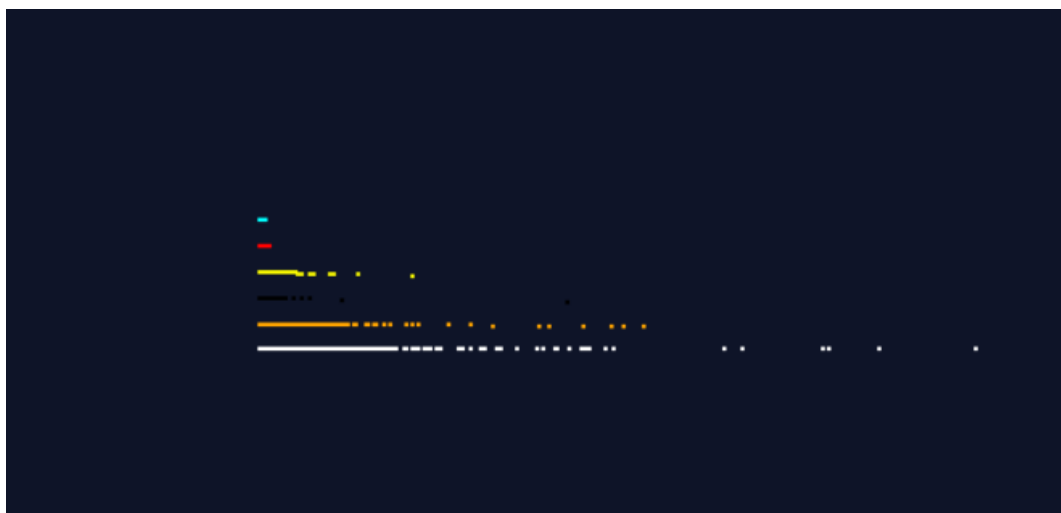
Jeden bod teda reprezentuje jedného zaregistrovaného užívateľa. Na X-ovú os mapujeme čas (smerom doprava rastie) registrácie, na Y jeho priemerné hodnotenie produktov (rastie smerom hore), Z-ovú os nechávame konštantne nulovú pre všetky body. Farby priradzujeme podľa pohlavia užívateľa, šedá je priradená užívateľom bez udaného pohlavia, červená ženám a modrá mužom. Z obrázku môžeme vypožorovať, že postupom času stále menej užívateľov udávalo pri registrácii svoje pohlavie, čo sa preukazuje prevažujúcou šedou farbou v pravej polovici grafu. Tiež vidíme, že tento trend nastal približne v polovici meraného časového úseku.

Ďalším vypožorovaným faktorom je, že postupom času začali pribúdať užívateľia, ktorých priemerné hodnotenie sa blíži minimu, čo je 0. Títo užívateľia sa rovnako rozhodli nezverejniť svoje pohlavie, z čoho môžeme usúdiť, že ide o falošné účty, ktoré sa zameriavajú na negatívne hodnotenie produktov.

Priezviská Aj súbor `surnames` obsahuje zaujímavé závislosti, my sme sa rozhodli demonštrovať rozdelenie priezvisk do skupín podľa etnika, v ktorom prevláda.

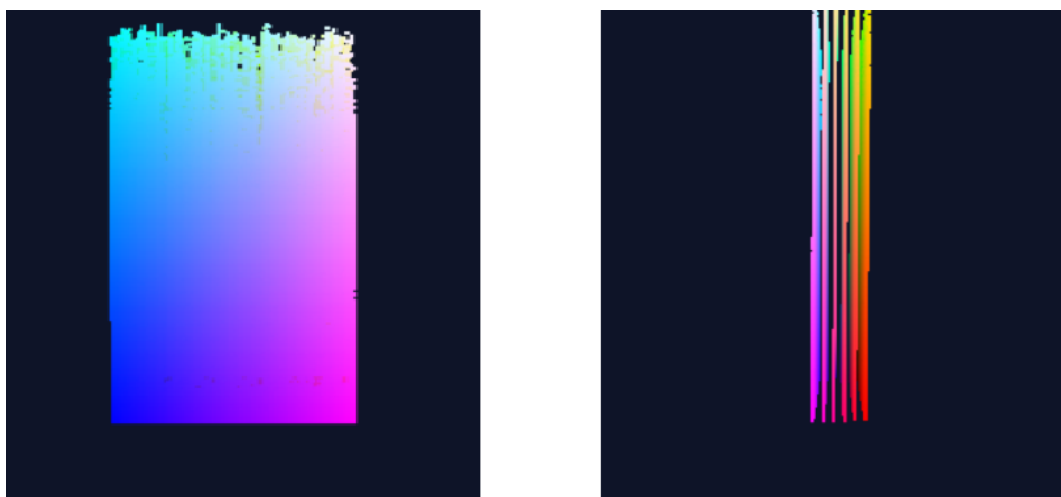
Každý bod predstavuje jedno priezvisko, na X-ovú os sme namapovali celkovú početnosť priezviska, Y-ovú os a farbu sme určili podľa rasového etnika farbou, ktorá je typicky týmto etnikám priradzovaná.

Na obrázku môžeme vidieť, že ľudia bielej rasy (mimo hispánskeho pôvodu) tvoria väčšinový podiel nositeľov najpočetnejších mien, avšak v prvej desiatke nájdeme aj mená, ktoré majú typický hispánsky pôvod. Tiež môžeme vidieť, že stále existujú priezviská, ktorých najpočetnejšími nositeľmi sú pôvodní obyvatelia Ameriky (červená farba), aj keď je počet týchto nositeľov nízky.

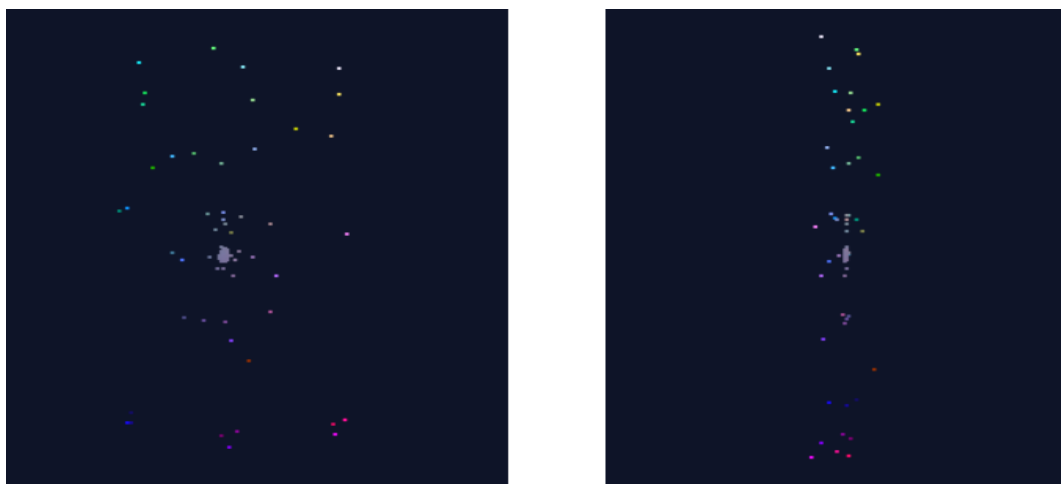


Ukážky zhlukovej analýzy Ďalším aspektom sledovaným vo vizualizácii je použitie zhlukovej analýzy na zredukovanie objemu dát. Bodmi, ktoré reprezentujú zhluky, sú ich geometrické stredy - centroidy. Na každej množine dát sme skúsili obe implementované algoritmy zhlukovej analýzy, teda Lloydov algoritmus a aj CCPD algoritmus.

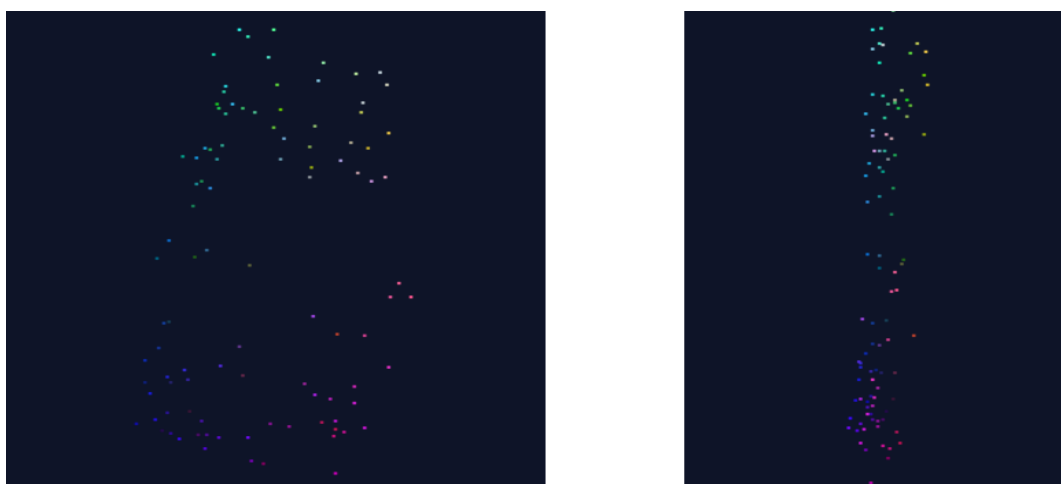
Vstupom algoritmov boli prvé 3 milióny záznamov zo súboru `evals`. Osi X odpovedá ID užívateľa, osi Y ID produktu a osi Z hodnotenie, rovnako sú všetky dimenzie využité na spojitú spočítanie farebných zložiek. Výstupné obrázky predstavujú pohľady na dáta z prednej a bočnej strany.



Obr. 5.2: Pohľad na mračno bodov bez modifikácie, počet vizualizovaných objektov presahuje 3 milióny.



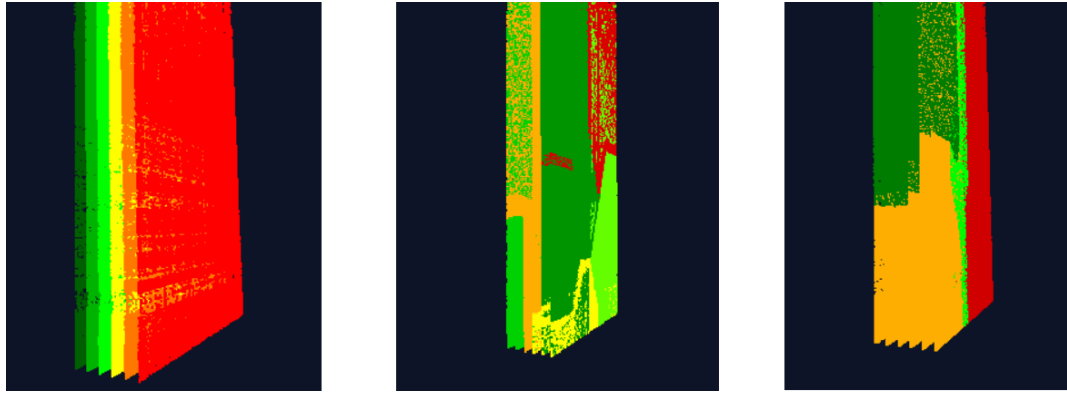
Obr. 5.3: Mračno bodov vyrobené Lloydovým algoritmom z datasetu *evals*.



Obr. 5.4: Mračno bodov vyrobené z rovnakého datasetu algoritmom CCPD.

Obidva algoritmy mali za úlohu vyrobiť 500 zhlukov, z ktorých stredov bola vyrobená nová množina dát. Pre algoritmus to znamenalo, že v jednej fáze muselo prebehnúť spolu 250 000 interakcií, čo znamená netriviálny nárast zložitosti jednej fázy oproti Lloydovmu algoritmu. Ten sme preto obmedzili oveľa väčším počtom fáz, aby oba výpočty prebehli v podobnom čase. Lloydov dostal limit 30 fáz, CCPD 5. Pri podobnom čase (CCPD 12,1 minúty, Lloyd 11,1 s využitím plného paralelizmu) dosiahol v tomto porovnaní algoritmus CCPD mierne lepšie výsledky, keďže takmer rovnomerne pokryl celú množinu pôvodných bodov, Lloydov algoritmus umiestnil príliš mnoho zhlukov do stredu množiny. Ukážkové obrázky nie sú plne objektívne, keďže inicializácia oboch algoritmov je založená na princípe náhody, avšak aj pri mnohonásobnom opakovaní pokusov sme dostali podobné výsledky.

Ďalšou úlohou algoritmov zhlučovacích algoritmov bolo vytvoriť menší počet zhlukov, opäť z veľkej množiny dát (prvé 3 milióny záznamov z *evals*, za účelom zafarbenia podľa príslušnosti k skupine. Metriky boli pre obe algoritmy rovnaké, diskrétna metrika pre identifikačné čísla užívateľov aj produktov, euklidovská pre hodnotenie.



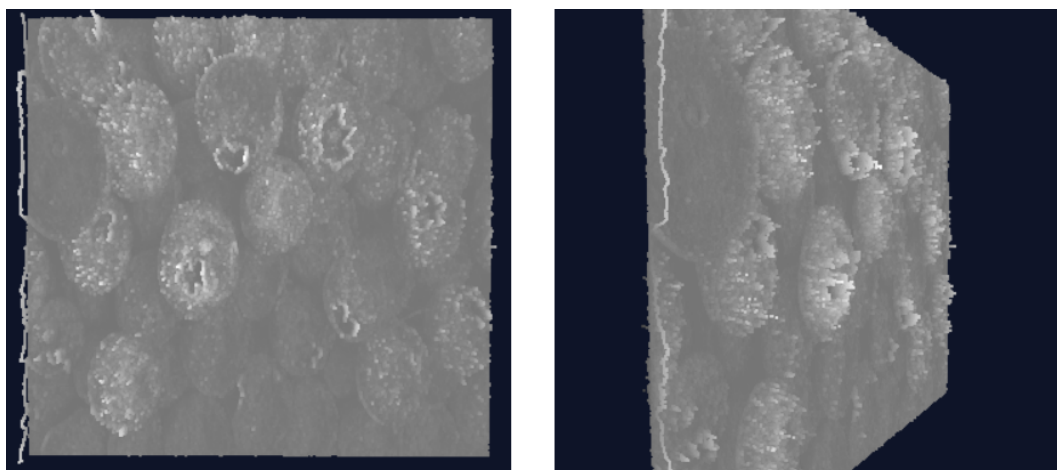
Na ľavom obrázku vidíme ofarbenie priamo podľa hodnotenia, ktoré predstavuje ideálne rozdelenie do množín. Parametre pre oba algoritmy boli rovnaké, limit maximálnych fáz rovný 100 a cieľový počet zhlukov rovný 6, výpočty prebiehali rovnako v podobných časoch okolo 3 minút. V strede vidíme dáta po aplikácii CCPD algoritmu, napravo výsledok Lloydovho algoritmu. Výsledky sú veľmi podobné aj po opakovanom spustení. Algoritmus CCPD ale zachováva veľkosť zhlukov, na čo sa Lloydov algoritmus neobmedzuje. Pre väčší počet fáz tak niektoré zhluky vo výpočte Lloydovho algoritmu môžu úplne zaniknúť, čo vedie aj vizuálne u niektorých farieb zastupujúcich zhluky k ich vymiznutiu.

Obraz Ako testovacie dáta obrazového vstupu sme vybrali voľne dostupné fotografie zo stránok www.pixabay.com. Na týchto fotografiách sme skúmali najmä možnosť zobrazenia výsledkov detekcie hrán v trojrozmernom priestore.

Prvou ukážkou je použitie filtra na detekciu horizontálnych hrán.

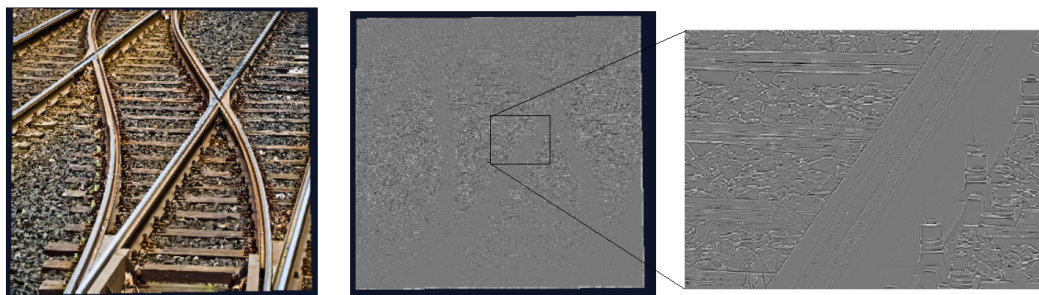


*Obr. 5.5: Vľavo: originál obrázku zobrazeného iba pomocou mračna bodov
Vpravo: Jasová zložka obrazu upravená filtrom detekcie horizontálnych hrán, hĺbka nie je použitá.*

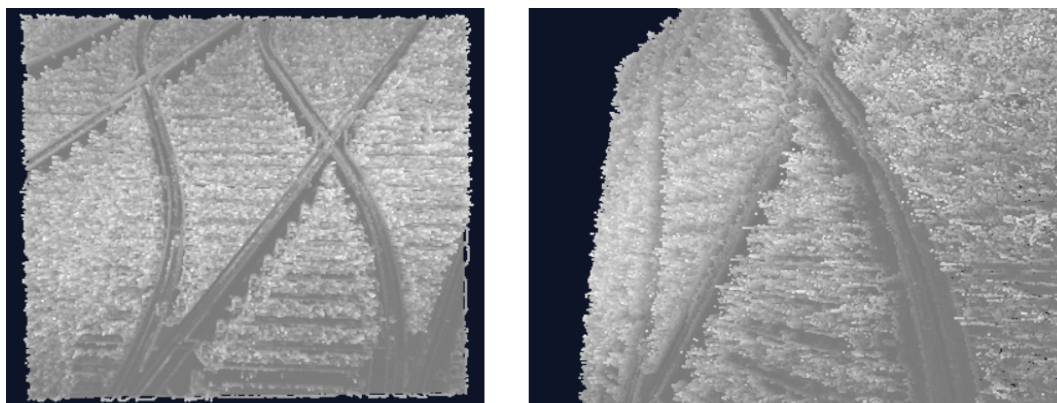


Obr. 5.6: Vľavo: Ukážka využitia tretieho rozmeru pri zobrazení obrázku po aplikácii filteru na detekciu horizontálnych hrán.
Vpravo: Rovnaká vizualizácia zobrazená z iného uhla demonštrujúca užitočnosť hĺbky.

Na druhý obrázok sme aplikovali filter detekcie hrán v oboch smeroch. Opäť sa ukázalo, že bez hĺbkového rozmeru je obrázok bez priblíženia málo názorný a hĺbka je v tomto ohľade veľmi užitočná.



Obr. 5.7: Vľavo: originálny obrázok
Stred: Jasová zložka obrázku po aplikovaní filteru.
Vpravo: Priblížený výsek obrázka ukazujúci, že hrany sú dobre viditeľné až po priblížení.



Obr. 5.8: Vľavo: Obrázok s mapovaním hĺbkového rozmeru na detekované hrany
Vpravo: Rovnaká vizualizácia zobrazená z iného uhla.

6. Uživatelská příručka

6.1 Pred spustením

Aplikácia je určená pre platformu Microsoft Windows. Kvôli tomu, že je projekt vyvinutý v prostredí .NET, je potrebné mať na počítači nainštalovanú podporu .NET Framework 4.5.2, ktorá je pre systémy Windows voľne dostupná. Všetky ďalšie externé knižnice sú dodávané spolu s aplikáciou.

6.2 Spustenie - vizualizačné okno

Po spustení sa otvorí okno vizualizácie, v ktorého strede na paneli (1) prebieha samotná vizualizácia. Hneď na úvod je možné na demo-vizualizácii kocky vyskúšať správne chovanie vykresľovania a tiež interakcie s vizualizáciou. Interakcia prebieha pomocou ľavého tlačidla myši a následného ťahu (otáčanie) a pomocou otáčania kolieskom myši (priblíženie).



Samotné nastavenie vizualizácie prebieha kliknutím na tlačidlo „Load new file“ (2) v ľavom dolnom rohu. Po kliknutí sa objaví okno nastavenia vizualizácie - Wizard. V prípade, že sme už súbor vizualizovali a chceme iba zmeniť parametre zobrazenia, môžeme kliknúť na tlačidlo „New visualization“ (3), ktoré použije naposledy používanú databázu a preskočí rovno na nastavenia vizualizácie - mapovania medzi záznamami databázy a ofarbenými objektmi.

Pod vizualizačným panelom je zašrtávacie políčko na ovládanie vertikálnej synchronizácie (4), napravo od neho nájdeme ukazateľ počtu zobrazených objektov (5), dolu vidíme výber farby pozadia (6). V pravom dolnom rohu sa nachádza aktuálny počet snímok za sekundu (7) a tlačidlo na obnovenie kamery do pôvodného stavu (8).

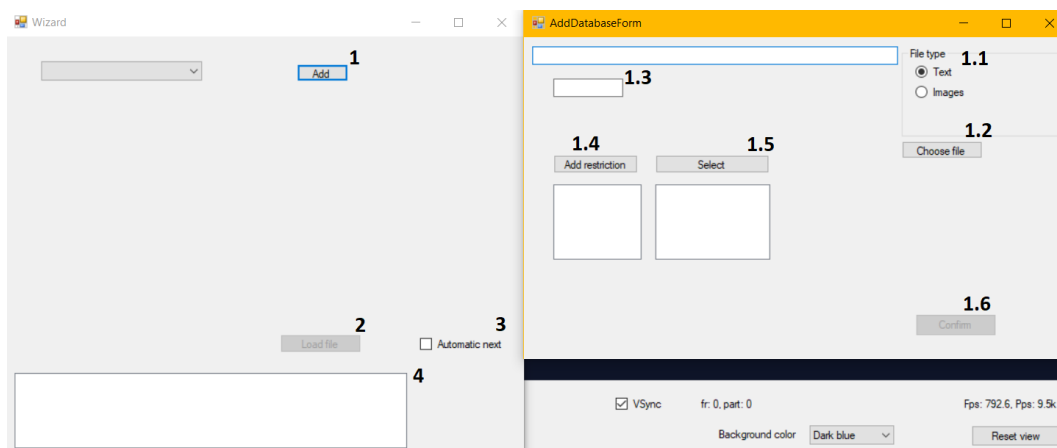
6.3 Okno nastavení - Wizard

Okno nastavení ponúka celú škálu možností, ako prejsť od datového súboru až k 3D modelu množiny dát. Jednotlivé operácie sú rozdelené do krokov podľa poradia v sekvencii spracovania. Medzi krokmi sa pohybujeme pomocou tlačidla „Next“, na každom kroku je možné nastaviť automatický prechod do ďalšej fázy po dokončení výpočtu pomocou zaškrtnutia „Auto-next“.

6.3.1 Načítanie súboru

Prvým krokom je výber zdrojovej množiny dát.

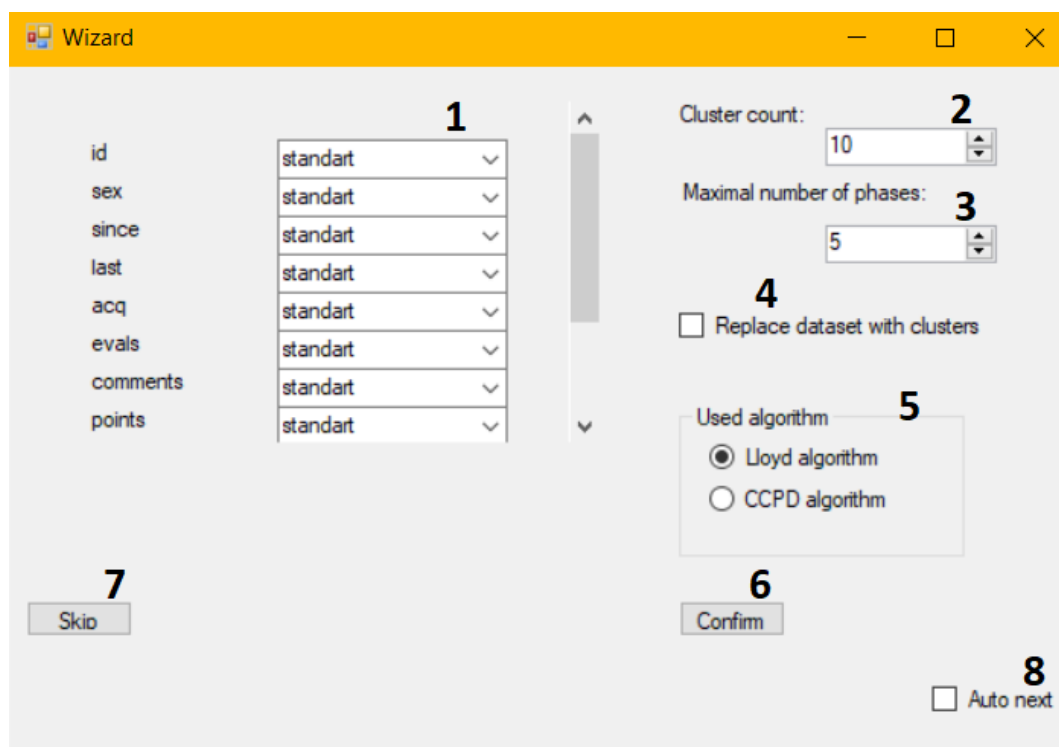
Základom je výber súboru, z ktorého sa dáta načítajú, tento výber je možný pomocou okna spusteného tlačidlom „Add file“ (1). Toto nové okno obsahuje samotný výber súboru - je potrebné špecifikovať typ súboru - text alebo obraz (1.1), a následne pomocou otvorenia systémového dialógu (tlačidlo „Choose new file“ (1.2) vybrať súbor. Tiež je možné aj zvoliť meno databázy (1.3) a konečne pravidlá uplatnené na súbor pri načítavaní. Takto je možné napríklad vybrať podmienku (1.4), ktoré musia načítavané dáta splniť, aby boli uložené do databázy, konkrétne podmienku na rovnosť alebo porovnanie, alebo je možné zvoliť selekciu atribútov (1.5), ktoré budú reprezentovať model dát a budú tak uložené v pamäti. V tejto časti je možné meniť aj názov a typ atribútu, podľa ktorého sa budú dáta čítať (v prípade nekompatibility medzi typmi hlási aplikácia chybu). Pomocou týchto dodatočných nastavení k súboru je možné vylúčiť nepotrebné záznamy alebo atribúty dát, čo vedie k menšej pamätovej náročnosti a zrýchleniu výpočtov. Následne je možné tieto nastavenia potvrdiť (tlačidlo „Confirm“ (1.6)) čím sa zavrie okno a vznikne záznam o novej databáze.



Po výbere načítanej databázy (výsuvný zoznam „Selected database“) nasleduje krok samotného načítania dát do aplikácie (tlačidlo „Load file“ (2)), pri čom po tejto fáze už nie je možné načítať žiadny ďalší súbor. Po započatí načítavania súboru je táto fáza ukončená, do ďalšieho kroku sa dostaneme pomocou zaškrtnutia políčka na automatické pokračovanie (3) alebo odpovedajúcim tlačidlom. Počas celého priebehu nastavovania vizualizácie je v dolnej časti prítomný panel správ (4), ktorým sú hlásené aktuálne informácie o pokroku operácií.

6.3.2 Transformácia dát

Zhluková analýza Ďalšou fázou je možnosť aplikovania algoritmov zhlukovej analýzy. K dispozícii sú zmeny metrick pre jednotlivé atribúty, počet žiadaných zhlukov (2) a limit počtu fáz algoritmu (3). Ďalej je na výber možnosť nahradiť celú množinu dát centroidmi spočítaných zhlukov (4), čím sa aplikuje zhluková analýza na redukcii objemu dát a samozrejme je aj možnosť vybrať použitý algoritmus (5).



Algoritmus je potom spustený pomocou tlačidla „Confirm“ (6). Tiež je ale ponúknutá aj možnosť celý tento algoritmus preskočiť (tlačidlo „Skip“ (7)). Celá táto fáza je preskočená v prípade, že vstupná množina dát obsahuje atribút, ktorý môže nadobúdať viacero hodnôt, keďže v takom prípade nie je možné definovať vzdialenosť medzi dvomi záznamami dát. K prechodu do ďalšej fázy slúži opäť tlačidlo „Next“ alebo príslušný Auto-next (8).

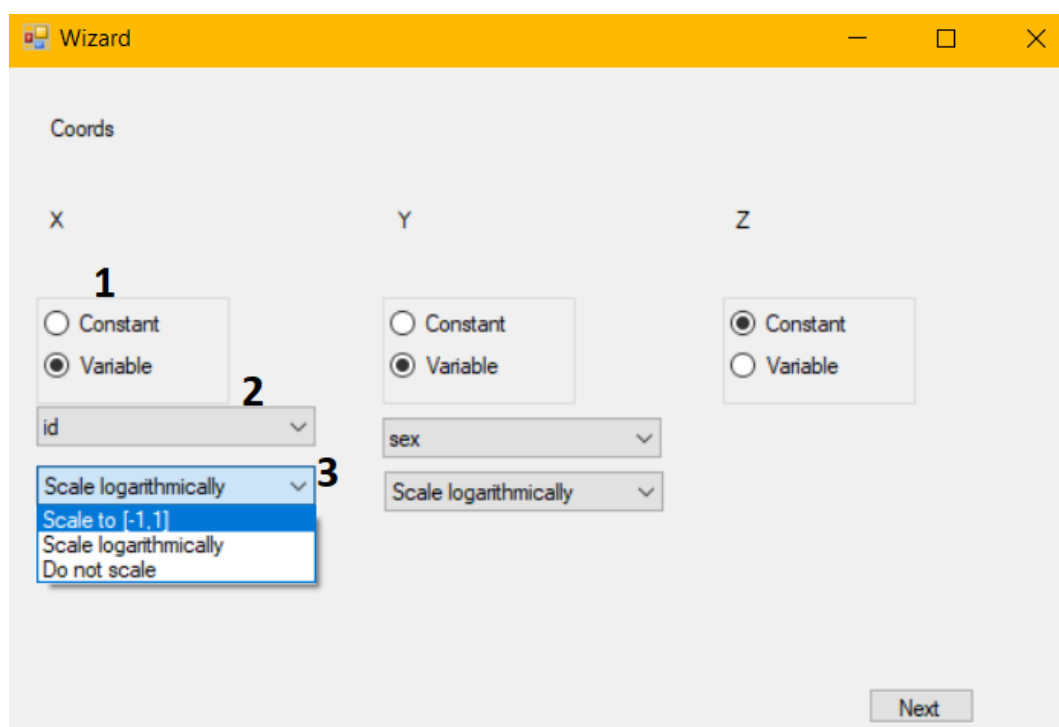
Konverzia Ďalším krokom je konverzia databázy do matice čísel. Táto podoba dát je potrebná v ďalšom priebehu transformácie aj vizualizácie, preto prebieha v tejto fáze. Konverzia je spustená hneď po prechode do tohto kroku, teda nie je potrebné túto fázu zapínať ani nič modifikovať.

Redukcia dimenzie Ďalšou možnosťou je transformácia dát pomocou algoritmu redukcie dimenzie - analýzy hlavných komponent. Jedinou modifikáciou je zmena počtu dimenzii, na ktoré budú dáta redukované. Pomocou tlačidla „Confirm“ je potom možné transformáciu spustiť. Túto fázu je samozrejme tiež možné preskočiť pomocou tlačidla „Skip“.

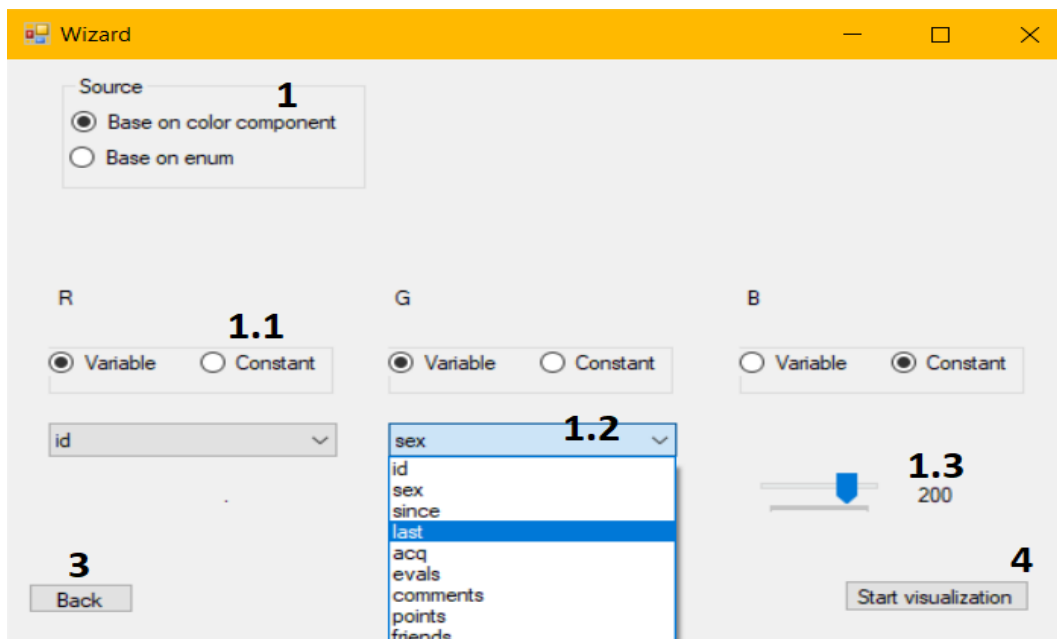
6.3.3 Vizualizácia dát

Poslednými dvomi krokmi vizualizácie je samotné nastavenie zobrazenia. Každý záznam z pôvodnej databázy je reprezentovaný objektom vo forme bodu v trojrozmernom priestore.

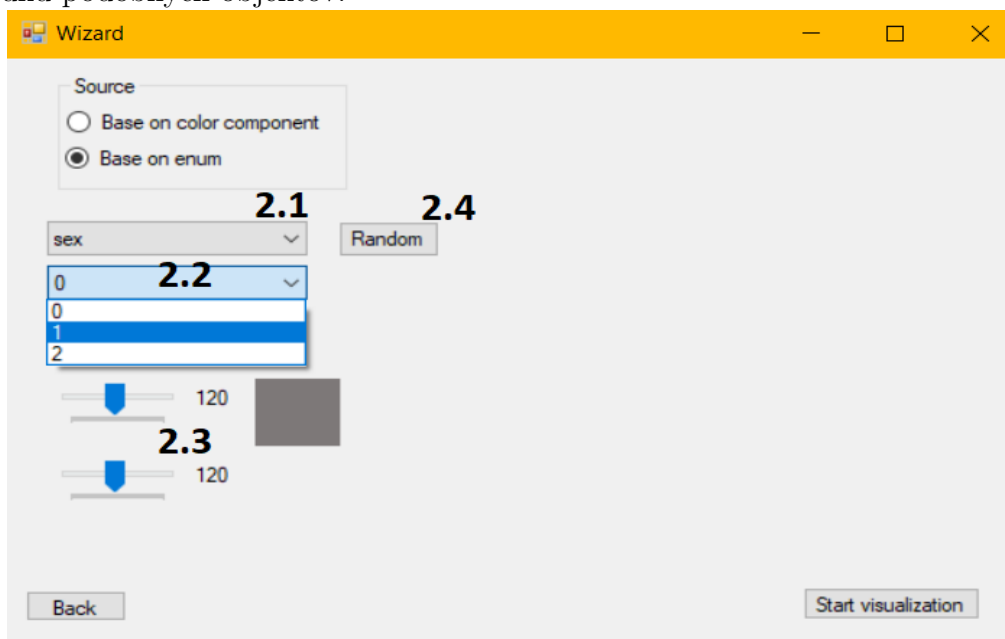
Prvým krokom je výber mapovania atribútov na súradnice objektu v trojdimenzionálnom priestore. Je možné rozmer vôbec nepoužiť, teda zvoliť, že všetky objekty budú mať hodnotu konštantnú alebo naopak vybrať variabilné hodnoty (1). Pre každý rozmer - X (šírka), Y (výška) a Z (hĺbka) je tak možné zvoliť atribút (2) a jeho škálovanie (3 - žiadne, logaritmické, škálovanie do jednotkovej kocky), na základe ktorého bude objekt v tomto rozmere umiestnený.



Druhým krokom je výber farby pre každý objekt. Pre farby sú ponúknuté 2 zdroje ofarbenia vyberané pomocou tlačítok v hornej časti (1). Podobne ako v prípade súradníc, aj tu je možné rozhodnúť pre každú zložku (varianta „base on color component“), či bude konštantná alebo variabilná (1.1), v tomto prípade môžeme priradiť jeden atribút (1.2), na základe ktorého bude zložka farby spojito mapovaná, alebo jednu konštantnú hodnotu posuvníkom (1.3). Iným prístupom k farbeniu dát je ofarbiť body na základe príslušnosti k určitej skupine (varianta „base on enum“), pri čom táto skupina je daná výčtovými hodnotami atribútov(2.1).



Je tak možné pre všetky hodnoty atribútu(2.2) vybrať farby buď pomocou posuvníkov (2.3), alebo pre každý vybrať farbu náhodne (tlačidlo „Random coloring“(2.4)). Keďže zhluky vyrobené zhlukovou analýzou sú očíslované indexami, pri čom pre každý záznam je táto hodnota indexu ochovaná v novom stĺpci databázy, užitočným využitím tejto varianty je ofarbenie pomocou príslušnosti ku zhluku podobných objektov.



Z panelu ofarbenia je možné sa ešte vrátiť tlačidlom „Back“(3) na panel súradníc a upraviť predošlé nastavenia. Po nastavení všetkých parametrov vizualizácie ostáva už len vizualizáciu spustiť (tlačidlo „Start visualization“). Po kliknutí na toto tlačidlo sa okno nastavení zatvorí a nastane presun vygenerovaných objektov do pamäte grafickej karty. Na základe počtu týchto objektov môže táto fáza trvať aj dlhší čas. Ak sú dáta úspešne presunuté, sú okamžite zobrazené v okne vizualizácie a užívateľ si ich môže interaktívne prehliadať.

7. Záver

7.1 Zhrnutie

Cielom tejto práce bolo vytvoriť aplikáciu, ktorá by poskytovala možnosti k transformácii a následnej vizualizácii veľkých dát.

Aplikácia bola vyvinutá v prostredí .NET, konkrétne v jazyku C#. Dôvodom tohto výberu bola prenositeľnosť medzi rôznymi procesormi, dobrá prehľadnosť kódu a neustále rozširovanie možností jazyka. Aplikácia ponúka aj vstavané užívateľské rozhranie, čo umožňuje okamžité použitie. Grafické prostredie WinForms sa ukázalo ako jednoduché, spoľahlivé a dostačujúce, v dnešnej dobe sa ale stále viac dostáva do popredia prostredie WPF, na ktoré by bolo možné projekt v prípade potreby previesť.

Ďalším cieľom bolo preskúmať štatistické metódy, konkrétne v oblasti redukcie dimenzie a zhlukovej analýzy a ich využitie v oblasti vizualizácie veľkých dát. Dospeli sme k záveru, že najvhodnejším algoritmom na redukciu dimenzie pre naše využitie je algoritmus Analýzy hlavných komponent. Oproti ďalším algoritmom vyniká najmä vďaka svojej pamäťovej a časovej nenáročnosti, čo je pri pohľade na objem očakávaných vstupných dát veľmi dôležité kritérium. Z tohto dôvodu sme ho aj úspešne implementovali v našom projekte.

V prípade zhlukovej analýzy sme rovnako vylúčili implementovanie časovo náročných algoritmov hierarchickej zhlukovej analýzy a implementovali sme rýchlejšie algoritmy na riešenie tohto problému a to konkrétne 2 varianty k-means zhlukovej analýzy. Ukázalo sa, že dosahujú dobré výsledky aj napriek svojmu charakteru optimalizačných metód.

Ďalším cieľom bolo preskúmanie možností využitia výkonu grafickej karty pomocou rozhrania OpenGL. Tento cieľ sme úspešne splnili a všetky záznamy dát je naša aplikácia schopná na základe predchádzajúcich nastavení vizualizovať interaktívne v trojrozmernom priestore.

7.2 Možné rozšírenia

Aplikáciu je možné rozšíriť vo viacerých ohľadoch, uvedieme pár príkladov.

Prvým možným rozšírením je doplnenie ďalších možností vstupu. V prípade textových dát by bolo možné doplniť zložitejšie parsery a umožniť tak načítanie rôznych ďalších formátov, napríklad aj súborov obsahujúcich grafické objekty .ply alebo .obj. V prípade veľkého množstva neštrukturovaných dát by boli užitočné aj parsery súborov .xml alebo .json, keďže tieto formáty sa dnes stále viac využívajú na reprezentáciu dát. Ďalšími zaujímavými vstupmi by mohli byť v budúcnosti napríklad aj videá alebo zvukové súbory.

Ďalším uvažovaným rozšírením by mohlo byť doplnenie štatistických metód slúžiacich na transformáciu dát. Aj napriek negatívnym predpokladom zložitosti by mohlo byť vhodné implementovať mnohé ďalšie algoritmy redukcie dimenzie alebo zhlukovej analýzy hlavne z dôvodu porovnania výsledkov na malých dátach, keďže už na stredne veľkých by výpočty trvali dlhý čas. Zlepšenie výkonu u týchto metód by sa mohlo dosiahnuť napríklad využitím "general purpose GPU", teda presunutím časti výpočtov na grafickú kartu.

V oblasti vizualizácie je tiež viacero možností zlepšenia, zaujímavou, avšak na implementáciu časovo náročnou úpravou by mohlo byť pridanie faktoru času a vytvorenie animovaných vizualizácií.

Zoznam použitej literatúry

- ABDI, H. a WILLIAMS, L. (2010). Principal component analysis. URL <http://www.utdallas.edu/~herve/abdi-awPCA2010.pdf>.
- BALZER, M., SCHLOMER, T. a DEUSSEN, O. (2009). Capacity-constraint point distributions: A variant of lloyd's method. URL https://kops.uni-konstanz.de/bitstream/handle/123456789/5934/Balzer_etal_2009_CCPDAVoLM.pdf.
- COX, M. a COX, T. (2008). Multidimensional scaling. In CHEN, C., HÄRDLE, W. a UNWIN, A., editors, *Handbook of Data Visualization*, pages 315–347. Springer-Verlag Berlin Heidelberg.
- DYM, H. (2013). *Linear algebra in action*. Second Edition. American Mathematical Society.
- JAMES, G., WITTEN, D., HASTIE, T. a TIBSHIRANI, R. (2013). *An Introduction to Statistical Learning*.
- KHRONOS. Khronos OpenGL Wiki. URL <https://www.khronos.org/opengl/wiki>.
- MORISSETTE, L. a CHARTIER, S. (2013). The k-means clustering technique: General considerations and implementation in mathematica. URL <http://www.tqmp.org/RegularArticles/vol109-1/p015/p015.pdf>.
- SHREINER, D., SELLERS, G., KESSENICH, J. a LICEA-KANE, B. (2013). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3*. 8th Edition. Addison-Wesley Professional.
- TAN, P., STEINBACH, M., KARPATNE, A. a KUMAR, V. (2018). *Introduction to Data Mining*, chapter Cluster analysis: Basic Concepts and Algorithms, pages 487–568. Second Edition. Pearson.
- WEISS, S. (2014). Gpus and GPU programming. URL http://www.compsci.hunter.cuny.edu/~sweiss/course_materials/csci360/lecture_notes/gpus.pdf.

A. Programátorská dokumentácia

Projekt je rozdelený do 3 logicky odlišných častí - časť užívateľského rozhrania, kde sa zaoberáme jednoduchým a prehľadným ovládaním programu z pohľadu užívateľa, časť datovú, kde riešime načítanie dát, ich efektívnu reprezentáciu z pohľadu pamätevej aj časovej zložitosti a aj ich transformáciu štatistickými metódami a nakoniec z časti vizualizačnej, kde zabezpečujeme správne zobrazenie dát na základe užívateľom zvolených nastavení.

A.1 Užívateľské rozhranie

Užívateľské rozhranie je založené na okennom prostredí Windows Forms (tiež aj WinForms) zloženom z viacerých knižníc. Toto prostredie je založené na systéme udalostí, teda všetky akcie sú spustené po vykonaní udalosti na strane užívateľa a obsluhou udalosti na strane programu. Prostredie WinForms obsahuje grafické prvky, na ktorých môže užívateľ vyvolávať akcie pomocou myši alebo klávesnice, ako napríklad okná, tlačidlá, zaškrťavacie políčka a posuvníky.

Na druhej strane obsahuje aj užívateľovi neviditeľné prvky, ktoré buď majú iba pomocné funkcie pre viditeľné prvky, ako napríklad panel, ktorý umožňuje programátorovi logicky zoskupiť grafické prvky, alebo majú samostatnú funkcionálnosť, napríklad časovač, ktorý pravidelne vyvoláva prednastavenú udalosť.

Rozhranie je zložené z dvoch hlavných okien - vizualizačného a okna nastavenia.

Vizualizačné okno

Vizualizačné okno je reprezentované triedou `VisualizationForm`, potomkom triedy `Form` z knižnice `System.Windows.Forms`. Hlavným komponentom je panel typu `GLControl`, definovaný v externej knižnici `OpenTK`, na ktorom prebieha samotná vizualizácia. Tento panel je obnovovaný na základe komunikácie s knižnicou `OpenGL`, ktorá zabezpečuje vykresľovaný obraz v 2D.

Pre panel sú tiež definované obsluhy udalostí, ktoré reagujú na pohyby myši a zabezpečujú tak zmeny parametrov zobrazenia (rotáciu a priblíženie) pomocou zmien v objekte triedy `Trackball`, na ktorý má okno referenciu. Objekt triedy `Trackball` obsahuje všetky parametre nastavenia kamere scény a tiež aj rozhranie, ktorými je možné toto nastavenie meniť. Parametre sú reprezentované trojrozmernými vektormi udávajúcimi polohu a smer pohľadu a štvorrozmernými maticami, ktoré predstavujú transformácie pohľadu, napríklad projekciu. Tieto položky sú potom predané rozhraniu `OpenGL`, ktoré ich použije v zobrazovaní.

Vizualizačné okno ďalej obsahuje základné štatistiky vizualizácie (počet snímkov za sekundu, počet zobrazených objektov) a tlačidlá na prechod k oknu nastavení.

Okno nastavení

Okno nastavení - Wizard - zhromažďuje ovládanie celkovej logiky programu. Predstavuje celý datový aj výpočetný tok, v tejto sekcii si preto popíšeme aj cel-

kový tok aplikácie. Samotné okno je rozdelené na oddelené logické časti, vizuálne zoskupené do panelov, predstavujúce kroky nastavovania vizualizácie.

Prvý krok obsahuje nastavenie vstupu - kliknutím na tlačidlo `Add database` nastane otvorenie nového okna zadávania vstupu `AddNewDatabaseForm`, kde sa vytvorí záznam o zadanom vstupnom súbore a na základe užívateľových preferencií sa pridajú nové reštrikcie na dáta, konkrétne na výber načítaných atribútov (analógia SQL Select, jednoduchá redukcia dimenzie) alebo na splnenie danej podmienky záznamom (analógia SQL Where, jednoduchá redukcia objemu dát). Všetky tieto nastavenia vstupu sa z tohto okna presunú do okna nastavení, kde je možné celý proces vstupu zopakovať a pridať tak ďalšiu databázu alebo proces výberu vstupu ukončiť a prejsť do fázy čítania vstupu (viac v sekcii Čítanie vstupu).

V druhom kroku je užívateľovi umožnené zapojiť do spracovania dát zhukovú analýzu. Základným parametrom je počet zhukov, ktorý implementované metódy potrebujú, ďalšou možnosťou je vybrať metriku, ktorá sa bude pri porovnávaní podobnosti záznamov v zhukovacích algortmoch používať pre jednotlivé atribúty, pri čom táto metrika je predaná ako referencia na metódu typu `delegate` do objektu stĺpca. Algoritmus zhukovej analýzy je možné po tomto nastavení spustiť alebo celý preskočiť.

Tretí panel slúži na sledovanie postupu konverzie zo stĺpcovo orientovanej databázy na riadkovo orientovanú vo forme dvojrozmernej matice reálnych čísel.

Po konverzii databázy do formátu matice čísel je možné aplikovať ďalšiu štatistickú metódu, redukciu dimenzie pomocou algoritmu PCA v triede `DimensionReduction`. Ak tento krok užívateľ nepreskočí, preložená databáza sa nahradí jej redukovanou variantou. Redukcia dimenzie je poslednou fázou transformácie dát pred vizualizáciou.

Posledné 2 panely sa venujú nastaveniu vizualizácie samotnej. Všetky parametre vizualizácie sú zabalené pre budúce použitie v objekte triedy `Settings`, ktorej inštancia je priradená oknu nastavení. Rozhranie triedy `Settings` obsahuje verejné metódy, ktoré slúžia k modifikácii a získaniu parametrov.

V prvej fáze užívateľ vyberá mapovanie medzi atribútmi a súradnicovým systémom vizualizácie, každému z rozmerov môže priradiť jeden atribút alebo nastaviť, že bude konštantný. Druhá časť sa venuje farebnosti, kde odlišujeme 2 druhy zafarbenia bodov - na základe jedného atribútu databázy, kde môže byť každej hodnote atribútu priradená jedna farba a všetky záznamy s touto hodnotou budú mať vo výslednom mraku bodov túto farbu, alebo je možné namapovať atribúty k farebným zložkám podobne ako to je u súradníc.

A.2 Vstup a reprezentácia dát

Akákoľvek hodnota, ktorá v aplikácii predstavuje hodnotu nejakého atribútu je v rôznych fázach spracovania a vizualizácie reprezentovaná pomocou iných štruktúr. V prípade, že je potrebné sa na dáta pozeráť riadkovo orientovane, teda že sú zložené zo záznamov, ktoré sú zoznamom heterogénnych hodnôt, napríklad hneď po načítaní zo vstupu alebo v dobe transformácie záznamu, sú hodnoty obalené do obalovacej triedy `Datum<T>`, ktorá je generická a dokáže tak obsiahnuť dáta ľubovoľného typu. Hlavným dôvodom je najmä požiadavka na zhromaždenie

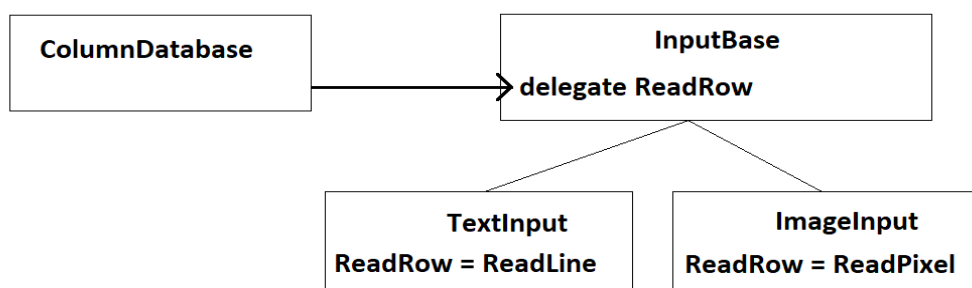
v jednom zozname rodičovského typu `DatumBase`, ktorý tak umožní zhromaždiť ľubovoľné dáta do jedného zoznamu a spracovávať a prenášať ako celok.

K reprezentácii celého záznamu dát, obalujúceho zoznam hodnôt atribútov, slúži trieda `RowRecord`. Keďže ide o zoznam, jeho dĺžka nie je obmedzená, jednotlivé hodnoty sa pridávajú na koniec metódou `AddValue`. Rozhranie triedy ďalej tvorí indexer pre získanie aj modifikovanie položiek a generická metóda `Get`, ktorá vráti rovno vybalenú hodnotu pre zadaný index.

Vstup

Základnou triedou, ktorá sa stará o získavanie dát je abstraktná trieda `InputBase` definujúca rozhranie každého vstupu, ktorý bude implementovaný. Obsahuje zoznamy mien a typov atribútov, ktoré budú predané nadradenej databáze pri vzniku úložiska, tiež aj hodnotu výčtového typu pre stav vstupu a nakoniec definuje typ delegáta `RowReader` a datovú položku tohto typu `ReadRow` vracajúci záznam dát `RowRecord`, ktorý tvorí základné vonkajšie rozhranie pre získavanie záznamov, ale nedefinuje jeho základnú implementáciu. Každá trieda, ktorá implementuje nejaký druh vstupu a dedí tak od triedy `InputBase`, musí `ReadRow` vyplniť svojou implementáciou generovania záznamov zo vstupu.

Všetky konkrétne vstupy sú konštruované pomocou statickej metódy `Construct`, ktorá pre zadaný súbor na základe jeho typu rozhodne, o aký vstup pôjde, vytvorí ho a vráti.



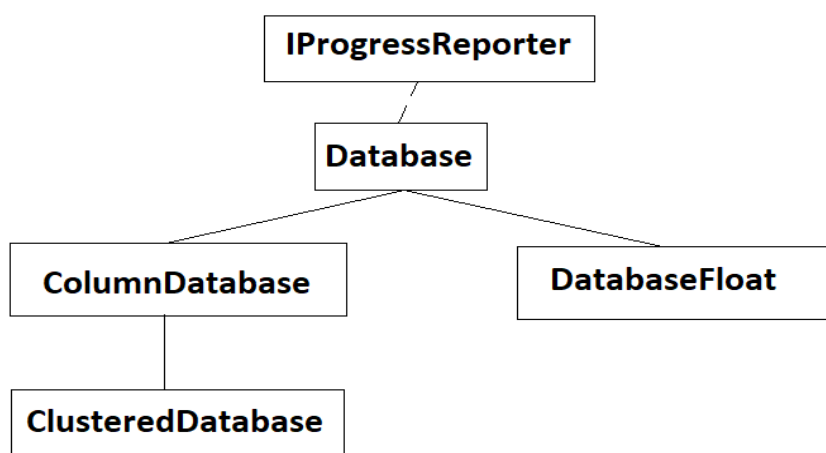
Implementáciou pre textové vstupy je trieda `TextInput`, ktorá je vytváraná pre súbory typu `.txt` a `.csv`. Súbor ale musí byť v správnom formáte „comma-separated value“, v opačnom prípade je vyhodnená výnimka. Vstup je čítaný po riadkoch, rozdelený na reťazce oddelené oddelovačom (‘;’), výnimkou, kedy sa neprihliada na oddelovače je oddelovač obsiahnutý v reťazci ohraničenom úvodzovkami (‘,‘), vtedy sú za hodnotu brané všetky ohraničené znaky. Každý záznam je kontrolovaný, či sa hodnoty v jednotlivých atribútoch zhodujú s príslušnými typmi stĺpcov, prípade nezahody je opäť vyhodnená výnimka.

Triedou zabezpečujúcou čítanie obrazových súborov je `ImageInput`. Prijíma súbory všetkých základných obrazových formátov a pomocou triedy `Bitmap` z balíka `System.Drawing` získava hodnoty farieb jednotlivých pixelov. Atribúty sú v tomto prípade vopred dané, každý záznam obsahuje súradnice, 4 zložky farebného modelu `RGBA` a 3 zložky modelu `HSV`. Pre tieto dva modely sme sa rozhodli, pretože `RGB` je základným a najpoužívanejším modelom v počítačovej

grafike a model HSV, ktorého hodnoty sú ľahko dostupné spočítaním z hodnôt RGB, je intuitívny z ľudského pohľadu - farbu charakterizuje intenzitou jasu, sýtosťou a odtieňom a je preto vhodným k vizualizácii práve týchto vlastností obrazu. Pre každý pixel je tak vytvorený práve jeden záznam exportovaný do databázy. Pre každý pixel vstupného súboru je tiež spočítaná hodnota jasovej zložky pomocou filtrov detekcie hrán.

Reprezentácia a uloženie dát

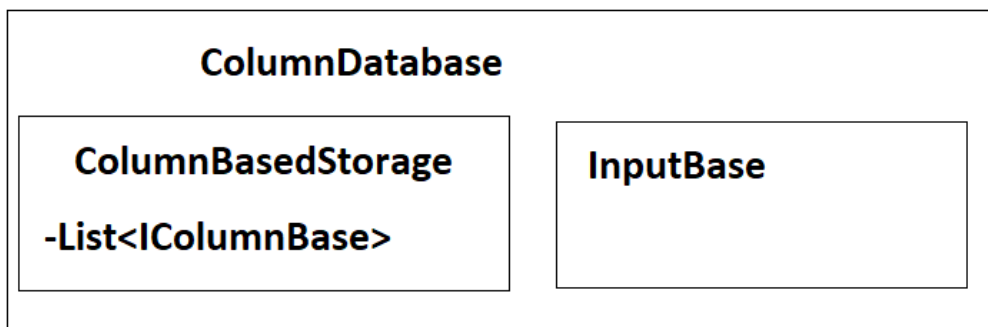
Dáta sú zhromažďované v objektoch tried s príponou Database, ktoré sú tiež aj potomkovia abstraktnej triedy Database. Napriek názvu nejde o pomocnú triedu k externému úložisku typu SQL alebo NoSQL databázy, všetky dáta sú v priebehu spracovania aplikáciou v manažovanej pamäti prostredia .NET, čo síce znemožňuje niektoré pokročilé operácie nad dátami, ale dosahuje vyššej rýchlosti odozvy.



Trieda Database definuje kontrakt, ktorý musia všetky dedičné triedy splniť a to napríklad vrátenie zoznamu názvov a typov atribútov, metódu GetValue na získanie zaobalenej hodnoty z indexu riadku a stĺpca databázy a rozmery databázy, či už počet záznamov, alebo atribútov.

Trieda ColumnDatabase Základnou implementáciou databázy v aplikácii je stĺpcovo orientované úložisko typu ColumnDatabase, objekt tejto triedy je vytvorený pre každý vstupný súbor a ukladá si všetky dáta zo vstupu. Základnou vlastnosťou tejto triedy je vlastníctvo kolekcie stĺpcov obalovacím objektom ColumnBasedStorage.

Samotné stĺpce sú reprezentované triedami, ktoré implementujú rozhranie IDatabaseColumn. Súčasťou tohto rozhrania je čítanie a zápis položiek na pozíciu danú indexom riadku, ako aj pridávanie na koniec stĺpca, každý stĺpec tiež musí vedieť vrátiť meno a typ atribútu, ktorý je v ňom uložený a nakoniec je v kontrakte obsiahnutý aj výpočet vzdialenosti medzi dvomi hodnotami, keďže tento výpočet nezáleží na type stĺpca, ale na preferenciách užívateľa, ktorý môže zvoliť ľubovoľnú metriku pre daný typ.

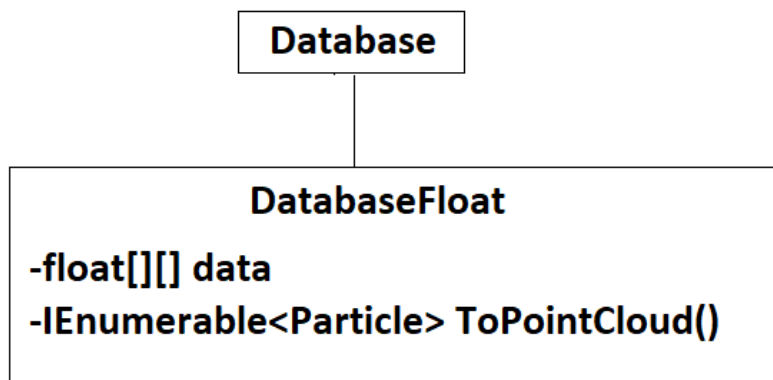


Stĺpce sa rozlišujú na 2 typy - hodnotové a referenčné. Hodnotové stĺpce, využité pre číselné typy atribútov, sa vyznačujú tým, že obsahujú priamo zoznam hodnôt. Do tejto skupiny patria triedy `IntDatabaseColumn`, `ByteDatabaseColumn` a `FloatDatabaseColumn`. Referenčné, použité pre reťazce, na druhú stranu ukladajú všetky svoje hodnoty do úložiska a v zozname reprezentujúcom stĺpec majú iba celočíselné indexy do tohto úložiska. Pre reťazce (`StringDatabaseColumn`) a skupiny reťazcov (`MultiStringDatabaseColumn`) je toto využitie úložiska výhodné, keďže každý reťazec, nezáleží na tom, aký je dlhý, je uložený iba raz a viac krát sa uloží iba celé číslo reprezentujúce index. Skupiny reťazcov nie sú ukladané spolu, ale každá časť je uložená samostatne, ich indexy sú potom zhromaždené do zoznamu a uložené do stĺpca. Výhodou je pamäťová úspora a tiež aj zrýchlenie overenia rovnosti hodnôt u dvoch záznamov, kedy nám stačí porovnávať indexy v úložisku namiesto porovnávanía reťazcov samotných.

Trieda `ClusteredDatabase` Trieda `ClusteredDatabase` je potomkom vyššie zmienenej `ColumnDatabase` a líši sa iba tým, že pridáva systém rozdelenia záznamov do zhlukov a dodatočnú funkcionálnu možnosť aplikovať algoritmus zhlukovej analýzy. Je vždy vyrobená z objektu triedy `ColumnDatabase` a to z toho dôvodu, že preberá všetky objekty stĺpcov naplnené datami z dôvodu úspory času a pamäti. Pridáva jeden dodatočný celočíselný stĺpec, ktorý označuje zaradenie záznamu do zhľuku pomocou indexu zhľuku.

Zhľuky sú tiež v dobe algoritmov reprezentované samostatným objektom `Cluster`, ktorý obsahuje množinu indexov záznamov, teda ide o spojenie opačným smerom, čo sa v algoritmoch ukazuje ako časovo výhodnejšie. Vo výsledku, teda v čase konverzie a vizualizácie je však lepšia reprezentácia stĺpcom.

Trieda `DatabaseFloat` Trieda `DatabaseFloat` predstavuje databázu určenú na vizualizovanie a ponúka iný pohľad na dáta ako stĺpcovo orientovaný, totiž dáta prekladá do čísel a ukladá v dvojrozmernej matici. Číselné atribúty sú uložené priamočiaro, z reťazcov odpadá uloženie konkrétnej hodnoty, ukladá sa len index do úložiska z pôvodnej stĺpcovej databázy. Vo fáze vizualizácie totiž nezáleží na hodnote reťazca, záleží len na odlišení reťazcov od seba, na čo index postačuje. Je tiež dedičnou triedou triedy `Database`, takže obsahuje informácie o názvoch a typoch dát a používa ich pre nastavenie vizualizácie v okne nastavení.



Na druhej strane ponúka aj rozhranie pre prenos dát smerom ku grafickej karte a možnosť transformovať všetky záznamy na základe nastavení vizualizácie do štruktúry `Particle`, čo sú objekty v trojdimenzionálnom priestore, ktoré majú priradenú veľkosť a farbu. Na to slúži metóda `GetGraphicObjectsFromData`, ktorá vracia všetky záznamy lazy-vyhodnocovaním. Tvorí tak poslednú zastávku dát, keďže tieto objekty sú bez medziukladania priamo zapisované do pamäte grafickej karty.

A.3 Transformácia dát

Transformáciu dát v našom projekte pomocou implementovaných štatistických metód zabezpečujú triedy reprezentujúce algoritmy, pričom metódy zhlukovej analýzy implementujú rozhranie `IClusteringAlgorithm` a metódy redukcie dimenzie rozhranie `IDimensionRedukcionAlgorithm`.

`IClusteringAlgorithm` vyžaduje implementovať metódu `PerformAlgorithm`, ktorá berie ako argument objekt typu `ClusteredDatabase` a každá implementujúca trieda by mala vstupnú databázu svojim výpočtom modifikovať. Implementáciami sú `text` triedy `LLoydAlgorithm` transformujúci databázu pomocou k-means algoritmu s Lloydovou metódou výmeny bodov a `CCPDAlgorithm` využívajúci ako metódu výmeny kapacitne obmedzenú distribúciu.

A.4 Vizualizácia

Celá logika vizualizácie je zhromaždená v triede vizualizačného okna. K ovládaniu rozhrania OpenGL je použitá externá knižnica OpenTK a statické metódy na triede `GL` v nej definované.

Pri spustení vizualizácie sa z vyrobenej databázy typu `DatabaseFloat` lenivým vyhodnocovaním vytvoria pre každý záznam objekty typu `Particle`. Štruktúra `Particle` obsahuje pre vizualizáciu dôležité dáta, a to polohu objektu v priestore, farbu a veľkosť. Po tom, čo sa pomocou rozhrania vytvorí `VertexBufferObject` a získa sa virtuálna adresa, ktorá je namapovaná na pamäť tohto objektu v grafickej karte, sú všetky získane objekty zapísané do grafickej karty podľa vopred definovanej štruktúry (zmenenej v kapitole 4.2.2).

Samotné vykresľovanie prebieha od tohto momentu v cykle, ktorí končí až vypnutím programu alebo nastavením ďalšej vizualizácie. V každom kroku tohto

cyklu sa naviažu hodnoty z VBO na vstupné parametre prvého shadera vo vykresľovacom reťazci (v našom prípade **vertex shader**), spočítajú sa konštantné vstupné parametre a spustí sa vykresľovací reťazec.

Shadere

Programy napísané v jazyku GLSL využívané vo vykresľovanom reťazci sú uložené v samostatných súboroch umiestnených v rovnakej zložke ako je spustiteľný súbor, konkrétne sú to **vertex.glsl** a **fragment.glsl**.

Vo vertex shaderi je aplikovaná projekčná transformácia daná štvorrozmernou maticou a tiež je tu rozkódovaná farba a veľkosť vrcholu, ktoré boli pôvodne z kapacitných dôvodov komprimované do jedného štvorbytového celého čísla. Poloha je posunutá rasterizéru, ktorý vyrobí z vrcholu žiadny až väčšie množstvo fragmentov, veľkosť vrcholu je zmenená pomocou vstavanej premennej a farbu vrcholu rasterizér priradí odpovedajúcim fragmentom.

Fragment shader už zabezpečuje iba zapísanie farby do výstupného frame bufferu.