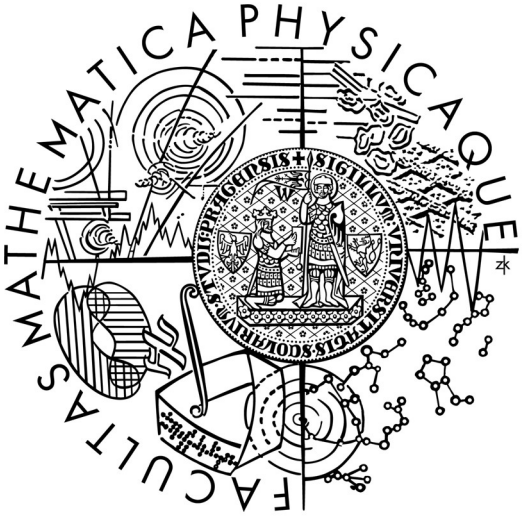
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Michal Halaša

Visualizátor struktury webu

Katedra softwarového inženýrství

Vedoucí diplomové práce:
RNDr. Leo Galamboš, Ph.D

Studijní program:
Informatika

Studijní plán:
Softwarové systémy

Poďakovanie

Chcel by som sa poďakovať RNDr. Leovi Galambošovi, Ph.D. za pomoc pri písaní tejto diplomovej práce.

Čestné prohlášení

Prohlašuji, že jsem svou diplomovou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze, dne 9. prosince 2009

Michal Halaša

vlastnoruční podpis

Obsah

Zoznam obrázkov

Zoznam tabuliek

Abstrakt

Název práce: Visualizátor struktury webu

Autor: Michal Halaša

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Leo Galamboš, Ph.D.

e-mail vedoucího: galambos@cythres.cz

Abstrakt: Webový robot generuje strukturu webu včetně popisků (anchor texty) příslušejících jednotlivým odkazům. Cílem práce je navrhnout a implementovat metodu visualizace tohoto orientovaného grafu. Implementace bude realizována v prostředí Java.

Implementace musí být schopna dynamicky měnit zobrazení v závislosti na zvolených kritériích - zobrazování pouze určitých odkazů nebo webových stránek. Zároveň musí být možné shlukovat skupiny odkazů nebo webových stránek pro zjednodušení zobrazované struktury.

Klíčová slova: webový robot, orientovaný graf, hyperbolická vizualizácia, java 3d

Title: Visualizer of the Web Structure

Author: Michal Halaša

Department: Department of Sofware Engineering

Supervisor: RNDr. Leo Galamboš, Ph.D.

Supervisor's e-mail address: galambos@cythres.cz

Abstract: Web crawler generates web structure including the notes (anchor texts) which belong to individual links.Thesis target is to analyze and implement method for visualisation of this oriented graph structure. Implementation language platform is Java.

Implementation must be able to denamically change the view according to defined criteria – showing only certain links or web pages. Grouping of links or web pages must be also available.

Keywords: web crawler, oriented graph, hyperbolic visualization, java 3d

1 Motivácia

Webový vyhľadávací robot Egothor[1] pri prechádzaní štruktúry webových stránok generuje indexové súbory, v ktorých je popísaná štruktúra odkazov (previazaní) medzi jednotlivými prehľadávanými stránkami. Cieľom tejto diplomovej práce je vizualizácia týchto dát pomocou grafovej reprezentácie na obrazovke počítača tak, aby bola možná dynamická práca a manipulácia zobrazovanej množiny dát. Jedná sa pritom o dáta s miliónovými počtami uzlov a hrán medzi nimi a súčasné zobrazovacie grafové algoritmy sú optimalizované pre stromy s maximálne 100.000 uzlami. Pri väčších počtoch prvkov grafu dochádza k dramatickému poklesu prehľadnosti a čitateľnosti informácii v zobrazovaných grafových štruktúrach.

Takáto vizualizácia pomôže pochopiť a analyzovať spôsoby previazania stránok medzi sebou. Tieto údaje môžu byť vysoko použiteľné pre dizajnérov a architektov webových riešení pri analýze dostupnosti a prístupnosti jednotlivých častí webu. Taktiež dokážeme zobraziť vstupné a výstupné odkazy webových stránok do iných domén, čo pomôže pri SEO[2] optimalizácii.

Cieľom je jednoduché a intuitívne zobrazenie v užívateľsky príjemnom prostredí desktopovej aplikácie na platforme Java bez neštandardných externých závislostí.

V prvej kapitole si popíšeme algoritmus H3, z ktorého sme vychádzali, potom v stručnosti predstavíme existujúci program H3 Viewer a jeho fungovanie, a nakoniec popíšeme, ako sme postupovali pri implementácii našej verzie algoritmu H3. Na záver si zhrnieme výsledky a vymenujeme otvorené problémy, ktoré by sa dali vyriešiť v budúcnosti.

1 http://www.egothor.org
2 Search Engine Optimization

2 Problematika zobrazovania grafov

Grafy majú prirodzený spôsob zobrazenia ako vrcholy a spápajúce hrany v priestore. Obrazová reprezentácia malých grafov je pre ľudí veľmi prirodzená. Keď myslíme na nejakú sadu informácii, predstavovanie si vzťahov medzi jednotlivými entitami pomocou grafov je veľmi pravdepodobné.

Napríklad v oblasti WWW sa na zobrazenie vzťahu stránok a podstránok na nejakej doméne (reprezentujúcich vrcholy) a odkazov medzi nimi (reprezentujúcich odkazy) používa skoro vždy grafové (často priamo stromové) zobraznie.

Oblasť teórie grafov nám ponúka priehrštie efektívnych algoritmov na prácu a spracovanie všeobecných grafov. Na druhú stranu, dve hlavné vedecké odvetvia zaoberajúce sa vizualizáciou dát sú počítačová grafika a oblasť psychológie zaoberajúca sa interakciou medzi počítačmi a ľuďmi nám dáva množstvo informácii o tom, ako ľudský kognitívny systém funguje a akým spôsobom spracuváva rôzne informácie.

Informačná vizualizácia je relatívne mladá vedecká (cca 25 rokov) podoblasť informačných technológii. Je to spôsobené tým, že až od doby približne polovice 90. rokov sa začínajú objavovať dostatočne výkonne zobrazovacie technológie, ako hardwareové (kvalitné grafické karty a monitory) tak aj softwareové (grafické knižnice, vyššie programovacie jazyky a hlavne postup v oblasti teórie grafov). V posledných rokoch sa jedná o veľmi živú a tvorivú oblasť vedeckého záujmu, čo je spôsobené bežnou dostupnosťou výkonných grafických systémov a veľkou praktickou využiteľnosťou dosiahnutých výsledkov.

Základný argument pre zobrazovanie dát je skutočnosť, že vizuálna reprezentácia dát pomáha ľudskému pozorovateľovi lepšie pochopiť štruktúru a obsah dát. Otázka vizuálnej reprezentácie je dokonca dôležitejšia pre tzv. informačnú vizualizáciu ako pre vedeckú vizualizáciu dát. Rozdiel

medzi týmito dvoma podoblasťami je hlavne v tom, že informačná vizualizácia hľadá priestorovú reprezentáciu dát, ktoré nemajú v sebe implicitne zahrnutý vzťah k priestoru (súradnice a veľkosť vrcholov, ich vzdialenosti atď). Narozdiel vedecká vizualizácia pracuje s priestorovými dátami, napríklad vzťahy medz atómami v molekulách, nervová sústava človeka alebo vesmírne objekty a vzťahy medzi nimi.

Vedecká vizualizácia je často používaná ako pomôcka pre nedokonalé ľudské zmysly tým, že pomáha zobrazovať a tým pádom pochopiť napríklad príliš rýchle (alebo pomalé) udalosti alebo príliš veľké (alebo malé) objekty, ktoré ľudské zmysly nedokážu priamo spracovať.

Na druhú stranu, typický súbor dát pre informačnú vizualizáciu je napríklad hustota obyvateľstva, kde by sa veľkosť mesta zobrazovala rôznou veľkosťou kruhového symbolu, alebo databáza kníh v knižnici, ktoré by sa dali rozložiť na časovú osu podľa dátumu vydania tak, že pre každý rok by na ose y bol vynesený počet kníh v tom-ktorom roku vydaný. Ako vidíme, výber priestorového zobrazenia je vhodný pre jeden typ dát ale nie pre iný. Súbor grafických elementov (vizuálne kódovanie) použitých na vizualizáciu by mal vždy zobrazovať sémantickú informáciu zobrazovaných dát, ktorú sa snažíme zobraziť.

Základom informačnej vizualizácie je priestorová pozícia a orientácia. Grafické elementy ako body, čiary alebo 3D objekty môžu byť posadené na tento základ. Tieto elementy často nesú doplňujúcu informáciu zakódovanú ako veľkosť, farbu, tvar, priehľadnosť atď. Podľa toho, aké atribúty sú vhodné pre aké typy dát, delíme zobrazované dáta do 3 skupín. Nominálne dáta sú navzájom rozlíšiteľné, ale nehodnotené (rôzne typy ovocia), ordinálne dáta majú prirodzené ohodnotenie (chemikálie s rôznym pH) a kvantitatívne dáta sú číselné, takže nemajú len poradie, ale aj vzdialenosť.

Efektívnosť premennej použitej na zobrazovanie závisí samozrejme na type dát, napríklad farebné odlíšenie je výborné pre nominálne dáta, ale nevhodné pre kvantitatívne dáta, veľkosť a dĺžka sú vhodné pre kvantitatívne dáta, ale nie pre nominálne dáta atď. Spoločným aspektom všetkých typov dát je to, že narozdiel od iných spôsobov zobrazenia, je priestorové (2D alebo 3D) zobrazenie

najvhodnejšie.

Základným problémom pri zobrazovaní siete vrcholov a hrán medzi nimi je konflikt veľkosti a vzdialenosti. Tzv. Gestalt princíp blízkosti hovorí, že vrcholy, ktoré sú zobrazené blízko seba, sú vnímané ako príbuzné, kdežto tie vzdialené od seba sú pre ľudské vnímanie nezávislé. Na druhú stranu atribút veľkosti má opačný vizuálny vplyv, takže väčšia vzdialenosť medzi objektami nám dáva lepší vzťah medzi ich veľkosťami. Prakticky všetky zobrazovacie algoritmy to riešia pomocou zvolenia si blízkosti narozdiel od veľkosti.

Niektoré nízko úrovňové vizuálne informácie sú zpracovávane automaticky ľudskými zmyslami bez vedomej pozornosti. Jedná sa o tzv. automatické alebo selektívne vnímanie a takto zpracovateľné atribúty sú dĺžka, smer, kontrast, tvar a farba. Toto automatické zpracovanie ale funguje skoro vždy len pre jeden atribút. Napríklad žltý objekt medzi čiernymi objektami je veľmi dobre viditeľný, ale žltý štvorec medzi červenými a zelenými štvorcami je ťažko rozpoznateľný.

Ďalšou využiteľnou vlastnosťou zobrazovania je vysoká oddeliteľnosť niektorých atribútov. Napríklad vzdialenosť a farba sú na sebe skoro nezávislé atribúty.

Využitím tohoto automatického (selektívneho) vnímania, môžeme uvoľniť ostatné spôsoby vnímania pre ďalšie úlohy, ktoré už vyžadujú vedomé zmyslové alebo rozumové zpracovanie.

Interaktívnosť je ďalším veľkým problémom pre počítačovú informačnú vizualizáciu. Objavením filmu, videa a hlavne počítačovej grafiky sa staré formy navigácie kompletne zmenili. Dynamická navigácia, jednoduché ovládanie a bezprecedentná výpočtová sila počítačov spôsobila revolúciu v spôsobe a princípoch interaktivity vizuálnych informácii.

Pojem interkativita má v rôznych kontextoch rozdielny význam. Dosť často je zamieňaná za animáciu, ale z pohľadu našich potrieb sa jedná v základe o tri oblasti. Prvou je navigácia, tj. možnosť zmeny uhla pohľadu alebo pozície v zobrazovanej scéne. Druhou oblasťou je možnosť

výberu. Jedná sa o schopnosť výberu elementov na scéne a taktiež úpravu scény pomocou vybraných nastaveni. Poslednou oblasťou sú animované prechody medzi jednotlivými stavmi scény, ktoré užívateľovi umožňujú prirodzený presun so zachovaním kontextu a adaptáciu na novú zobrazovanú scénu.

Najjednoduchším spôsobom interaktivity je imitácia reálneho sveta, tj. imitácia práce s papierom pomocou posúvania, približovania a odďaľovania. Typickým príkladom je práca s mapovými software. V prípade využitia 3D technológii je potrebné použiť algoritmy, ktoré pomocou rotácie, presunu, škálovateľnosti, perspektívy a následným premietnutím do 2D obrazovky počítača dokážu byť pre ľudské vnímanie stále pochopiteľné. K tomuto nám pomáha už spomínaná kognitívna psychológia, ktorá objasňuje spôsoby ľudského vnímania okolia a fungovania kognitívnej sústavy.

Počítače umožňujú posunúť hranice zobrazovania od jednoduchého zrkadlenia reálneho prístupu k menej priamym spôsobom pomocou tzv. deformačných techník. Jedná sa napríklad o techniku rybieho oka alebo premietanie do hyperbolického priestoru, kedy je užívateľovi daná možnosť zmeny miesta záujmu, na ktorom je mu poskytnutá možnosť prezerania scény na vyššej úrovni detailu.

Jednou z významných výziev vizualizačných systémov je, ako zobraziť čo najviac informácii na konečnej ploche danej obrazovkou počítača. V prípade, že zobrazované informácie sú príliš veľké (alebo malé) pre zobrazovanú plochu, tak pomocou technológie Pan&Zoom je užívateľovi umožnené pohybovať pomocou zobraziteľnej časti scény a takisto meniť vzdialenosť (mierku) scény. Takýto prístup ale má jednu zásadnú nevýhodu, a tou je strata užívateľovho pojmu o mieste v scéne, kde sa práve nachádza a ako sa tam dostal. Toto sa dá čiastočne vyriešiť pomocou malého podokna so zobrazovanou scénou v nejakom základnom priblíženi a približným umiestnením, kde sa užívateľ, v konkrétnom okamihu, nachádza.

Veľa algoritmov sa viac-menej úspešne snažilo vyriešiť tento problém so stálým prepínaním pozornosti užívateľa pomocou spomínaných deformačných techník tak, aby užívateľ aj pri vyššej úrovni detailu pri prezeraní scény dokázal mať prístupné všetky dôležité informácie o pozícii

v scéne na jednej obrazovke.

Na druhú stranu posudzovanie a hodnotenie vizualizačných systémov je omnoho zložitejšie oproti štandardným grafickým systémom preto, lebo je náročné objektívne posudzovať subjektívne vnímanie (ako bolo ukázané vyššie) zobrazovaného systému. Tiež je netriviálne posúdiť, či vizualizačný softvér naozaj pomáha k rýchlejšiemu dosiahnutiu požadovaných výsledkov ako bez neho. Na druhú stranu sa dá celkom jednoducho posúdiť, či nový software je rýchlejší a fotorealistickejší ako porovnávaný.

Ďalším aspektom vizualizačných algoritmov je veľkosť grafu, ktorý sú schopné zvládnuť. Malé grafy môžu byť vykreslené ručne (jednotky vrcholov), ale pomocou počítačov sa dá dosiahnuť užívateľsky prístupné zobrazenie grafov až s niekoľkými miliónami vrcholov. Pri takýchto veľkých grafoch ale čas potrebný na predprípravu vstupných dát do formy vhodnej (optimalizovanej) pre zobrazovací algoritmus sa stáva jedným z kritérii algoritmu, tak isto ako rýchlosť odpovede, tzv. responsivness. Napríklad algoritmus, ktorý dokáže zobraziť graf s 10 miliónmi vrcholov, ale predpríprava dát trvá jeden deň a po vykreslení je scéna neovládateľná, nie je pre väčšinu prípadov vhodný.

Ako vieme, veľa problémov teórie grafov je NP-úplných, a preto aj ich aproximácia býva často veľmi výpočtovo a časovo náročná. Väčšina grafových algoritmov sa zameriava nie na veľkosť vstupných dát ale na užívateľskú prítuľnosť následne zobrazenej scény, a preto sú takéto algoritmy vhodné len pre vstupné dáta do niekoľko sto vrcholov. Na prvý pohľad sa jedná o dostatočnú veľkosť vstupných dát pre bežné problémy, ale napríklad počty počítacov v sieti Internet idú do stoviek miliónov a počet webových stránok dosahuje podľa údajov firmy Google desiatky miliárd[3].

3 http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

3 Výber algoritmu

Pre porovnanie sme zvolili 4 vizualizačné algoritmy, ktoré zobrazujú hierarchické datové štruktúry, čo je aj náš prípad. Jedná sa o Walkerov algoritmus, Magic Eye, InfoLens a hyperbolickú vizualizáciu [3].

Porovnávali sme ich z pohľadu typu vykresľovaného stromu, podpory technológie Focus+Context, práce s veľkými grafmi a navigáciou.

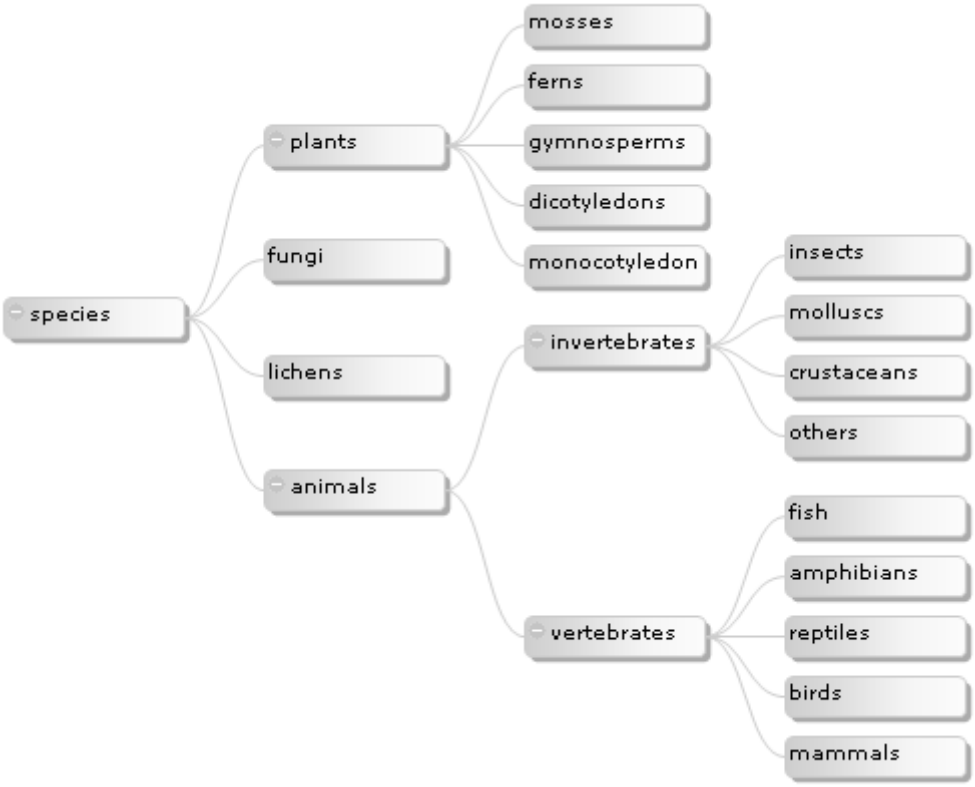
3.1 Walkerov algoritmus

Walkerov algoritmus je jednoduchý top-bottom algoritmus na vykreslenie stromu bez deformácie. Cieľom algoritmu je zobrazenie užívateľsky príjemného grafu. čo je dôležité pre ľudské vnímanie. Vrcholy a podstromy majú pridelený rovnaký zobrazovaný priestor a rodič je vždy vstrede nad svojimi potomkami. Jedná sa oveľmi intuitívne zobrazenie, ktoré sa používa prakticky ako štandard pre vizualizačné grafy s malým počtom vrcholov.

Jeho veľkou výhodou je lineárna zložitosť vzľadom na počet vrcholov.

Kým je graf dostatočne malý, tak užívateľ môže pochopiť celú štruktúru grafu a každý detail naraz. Ak sa jedná o stromy s väčším počtom vrcholov, tak pre každý z nich musí byť vyhradený menší priestor, takže sú jednotlivé vrcholy vykreslené veľmi blízko seba, čím sa stráca informácia o detailných informáciách v listoch. Na zlepšenie kvality detailu sa môže použiť technológia Pan&Zoom, čo ale zase prináša stratu celkového pohľadu na celú scénu, ako už bolo popisované vyššie.

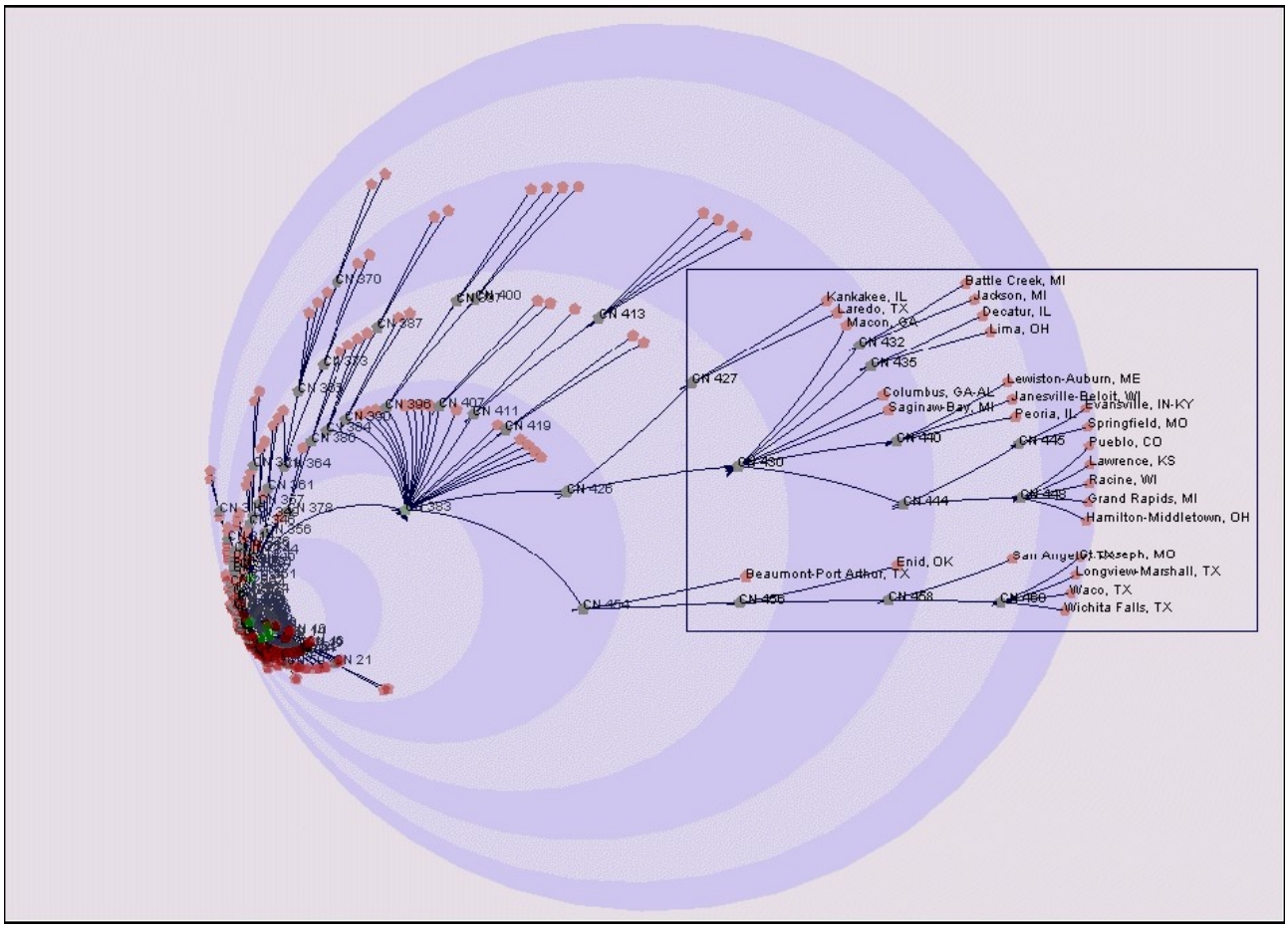
Dôležitosť Walkerovhoalgoritmu , na dnešnú dobu už zastaralého, je v tom, že všetky ostatné popisované grafové algoritmy z neho vychádzajú.



Obrázok 1: Walkerov algoritmus zobrazujúci rozdelenie zvierat do druhov

3.2 Magic Eye

Magic Eye rozloženie požíva hviezdicové vykresľovanie Walkerovho algoritmu na pologuľu. Tento algoritmus je vhodný pre väčšie grafy ako Walkerov algoritmus, ale stále je veľkým problémom konečný priestor (povrch pologule). Pomocou technológie Pan&Zoom a Focus+Context sa dá vyriešiť problém s nízkou úrovňou detailu pri okrajoch pologule, ale ztráca sa takto celkový pohľad na scénu.



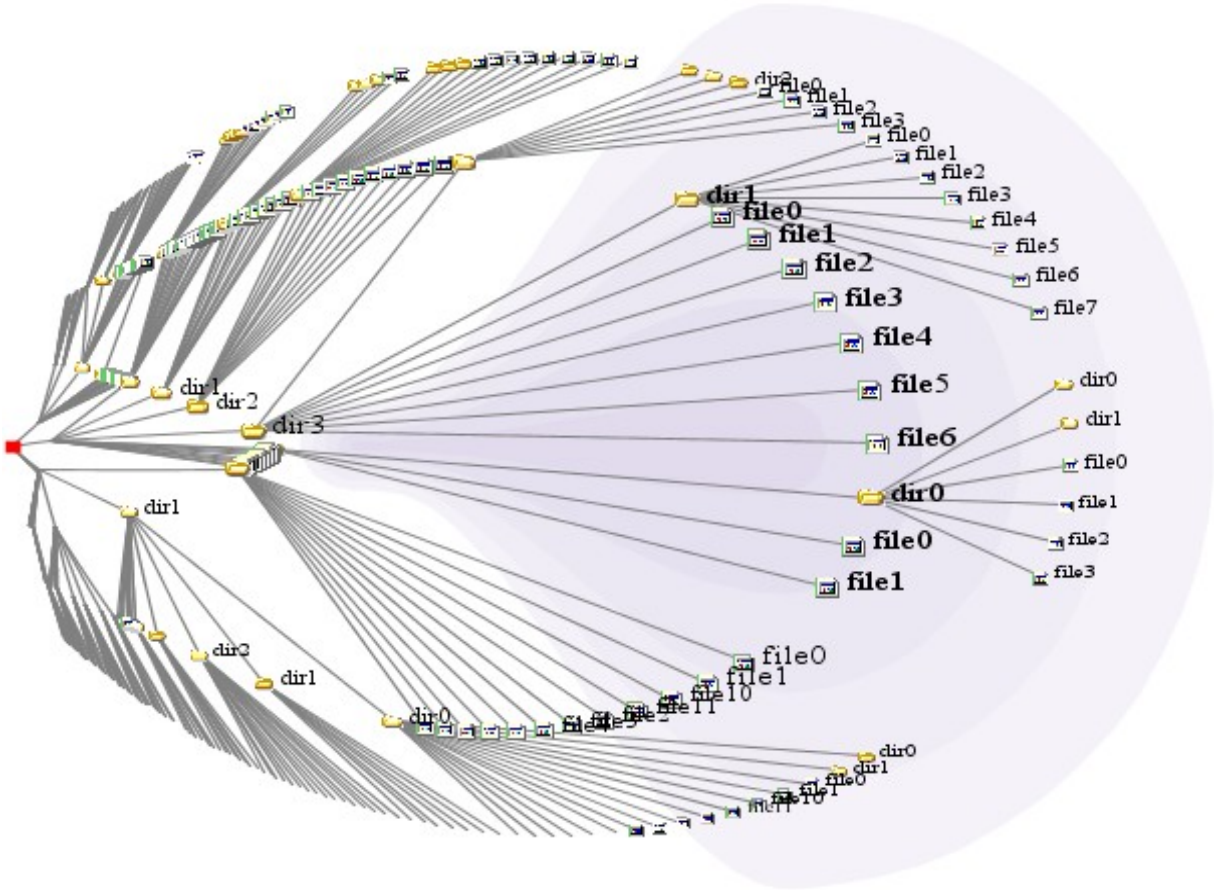
Obrázok 2: Magic Eye View so zaostrením na pravú časť projekčného kruhu

3.3 InfoLens

InfoLens algoritmus je vylepšený algoritmus Magic Eye. Naviac umožňuje, okrem zmeny uhla pohľadu, aj posun grafu po pologuli, na ktorú je premietaný a používa techniku FishEye deformácie pre dosiahnutie lepších možností približovania, intuitívnejšieho ovládania a tzv. fan-out efektu, ktorý je pre ľudské oko esteticky príjemný.

Fan-out (rozptyľovací) efekt odsúva pomocou plynulej animácie objekty, ktoré nie sú v priamom záujme na okraje zobrazovanej scény, čím sa zprehľadňuje vyššia úroveň detailu.

FishEye (rybie oko) deformácia zväčšuje oblasti v centre záujmu a ostatné oblasti sú zobrazené s nižším detailom. Rovnaký efekt sa dá dosiahnuť pri prezeraní objektu pomocou vypuklej šošovky. Pomocou tohto efektu je algoritmus InfoLens vhodný pre väčšie grafy ako Magic Eye, ale stále nedosahuje dobré výsledky pre veľké grafy.

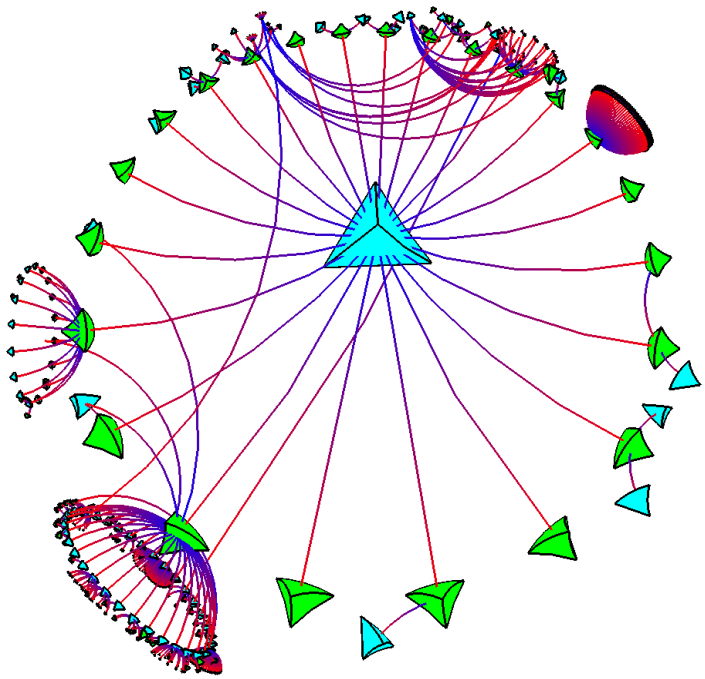


Obrázok 3: InfoLens algoritmus

3.4 Hyperbolický algoritmus

Hyperbolický algoritmus používa tiež hviezdicové vykresľovanie, takže priestor pridelený podstromom závisí na počtu vrcholov v ňom. Tento postup v Euklidovskej geometrii nie je tak efektívny vzhľadom na pokrytý priestor ako Walkerov algoritmus, avšak v tomto prípade je strom vykresľovaný v hyperbolickom priestore, ktorý poskytuje dostatok priestoru pre takéto vykreslenie. Exponenciálne rastúci priestor k polomeru je hlavnou výhodou tohoto prístupu.

Poincarého kruh (hyperbolický priestor premietaný do gule v euklidovskom priestore) poskytuje nekonečný priestor, takže celé stromy môžu byť vykreslené vnútri tohoto priestoru. Narozdiel od Walkerovho algoritmu nie sú stromy vykreslené rovnomerne (pri počiatku gule je použité malá mierka a pri okrajoch veľká). Tento nedostatok sa dá vyriešiť pomocou technológie Focus+Context.



Obrázok 4: Hyperbolická vizualizácia súborov na disku

3.5 Porovnanie vybraných algoritmov

Porovnanie sme zhrnuli do tabuľky, v ktorej sú zhodnotené vlastnosti týchto štyroch vizualizačných algoritmov pre stromové štruktúry.

	Walker	**Magic Eye**	**InfoLens**	**Hyperbolický**
Hviezdicové vykreslenie		x	x	x
Vhodnosť pre malé grafy	+	+	+	+
Viditeľnosť celej scény		x	x	x
Celkový pohľad	-	+	x	x
Focus+Context	-	+	+	+
Permanent focus area		-	+	+
Pan&Zoom	-	-	-	+
Fan out efekt	-	-	+	+
Vhodnosť pre veľké stromy	-	-	-	+

Tabuľka 1: Porovnanie vykresľovacích algoritmov

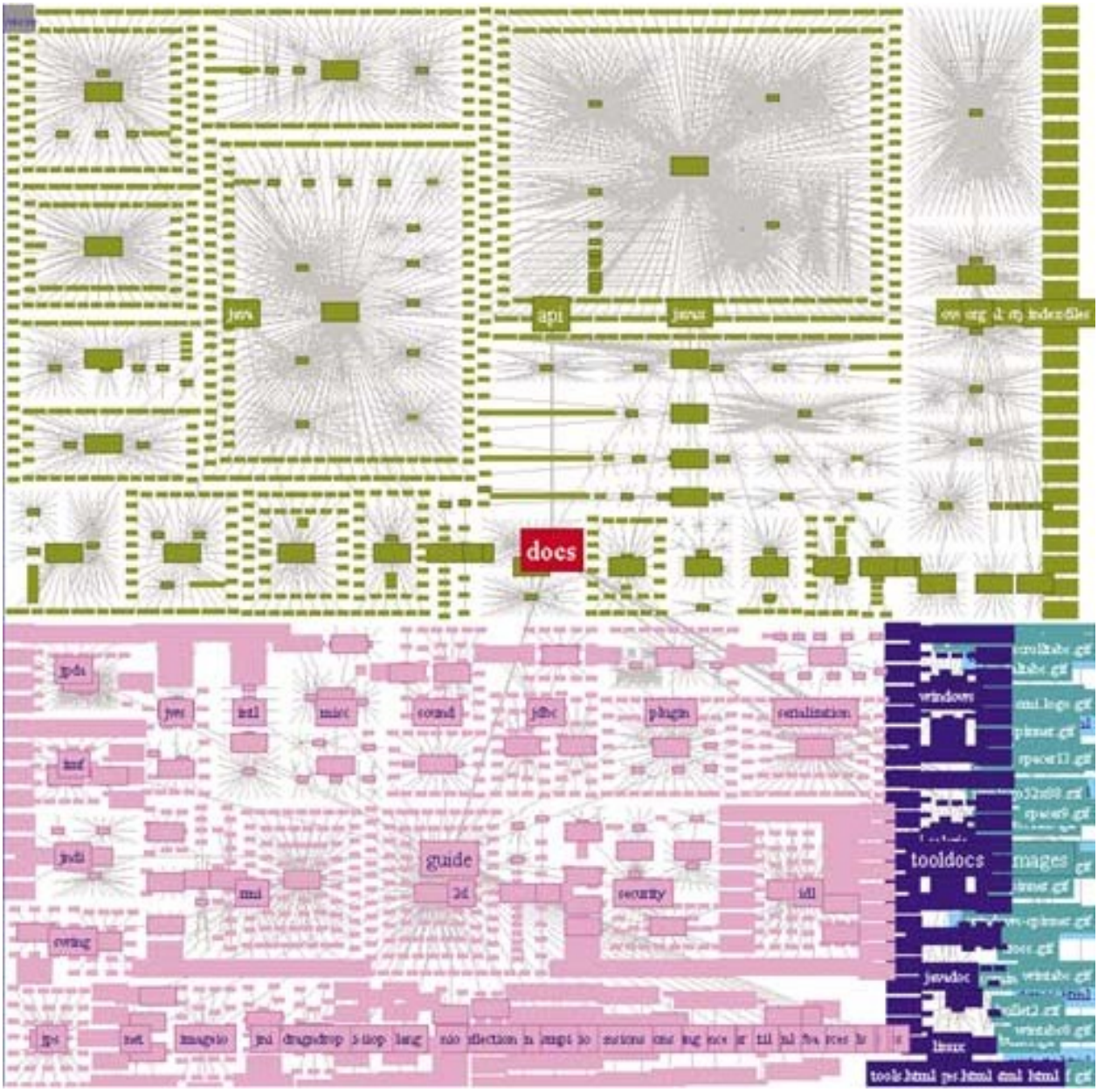
Legenda:

- + algoritmus vlastnosť má veľmi dobre použiteľnú
- x vlastnosť je použiteľná
- - existuje, ale bez veľkého prínosu
- v prípade, že nie je uvedené nič, tak algoritmus danú vlastnosť nemá

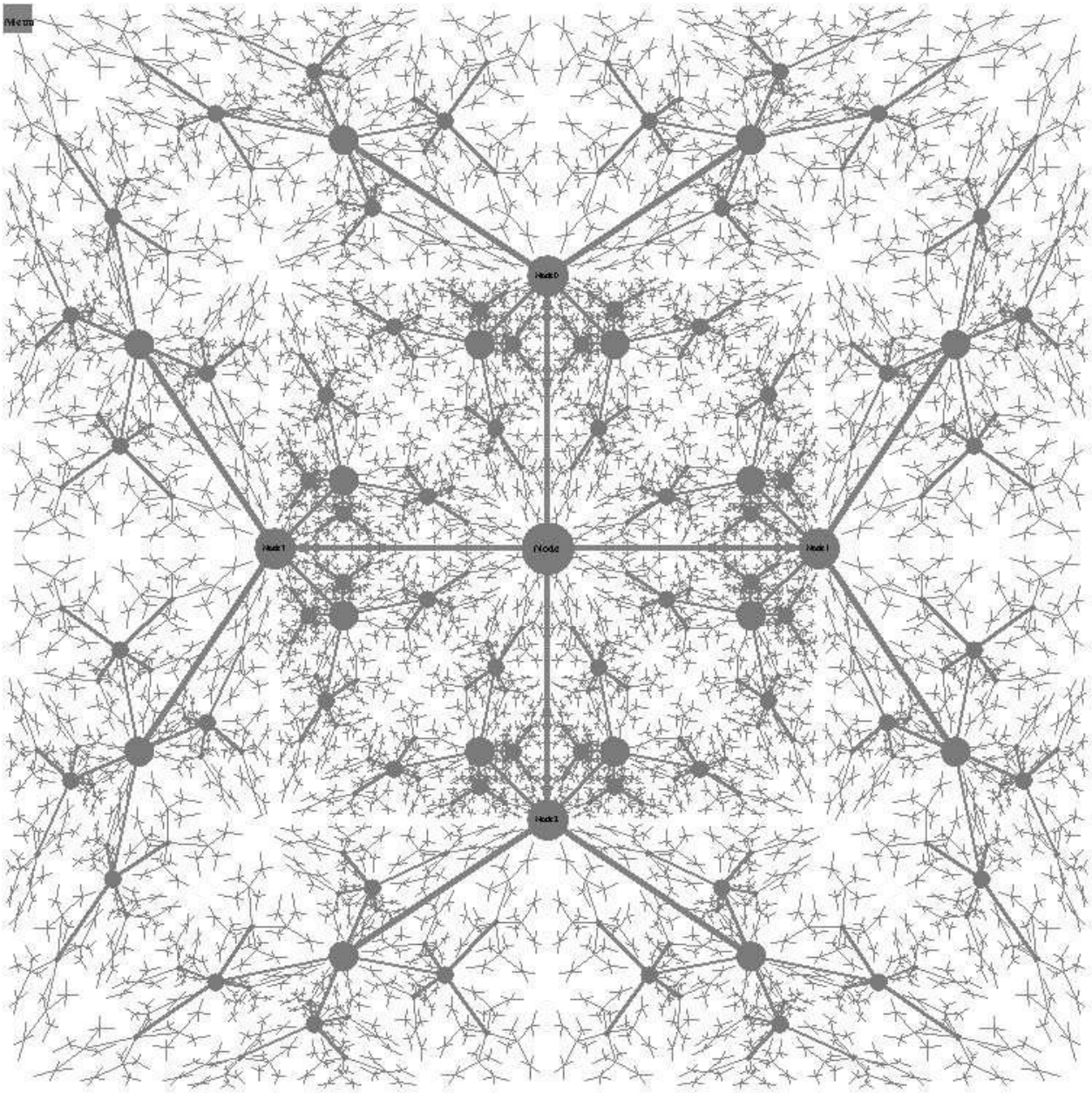
Samozrejme existujú aj iné algoritmy na vizualizáciu veľkých dátových štruktúr (niektoré z nich sú určené aj pre stromové štruktúry), ale buď ich vlastnosťou je veľmi špecializované rozloženie prvkov grafu (ako vidieť na Obrázku č. 5 a 6) alebo pojem *veľkých* dátových štruktúr obmedzujú na grafy s tisíckami, v lepšom prípade desiatkami tisíc, prvkov (vrcholov alebo hrán), čo je pre náš prípad viac ako nepostačujúce.

Hyperbolická vizualizácia sa ukázala byť veľmi intuitívna na pochopenie pre používateľa aj vhodná na zobrazenie hustých grafov (s množstvom spätných hrán), čo je prípad previazania webových stránok. Algoritmus H3 zase ponúka robustné riešenie pre veľké grafy s prepracovaným, efektívnym a hlavne rýchlym algoritmom stavby grafu.

Nakoniec sa táto voľba ukázala ako správna, pretože sa nám podarilo zobraziť aj husté grafy s niekoľkými miliónmi uzlov tak, že ovladánie a manipulácia grafu fungovala plynule a interaktívne na bežnom hardware.



Obrázok 5: Algoritmus EncCon (graf reprezentuje dokumentáciu k Java SDK v rozsahu 9500 podadresárov s 10000 súborov)[5]



Obrázok 6: 22000 vrcholov vykreslených pomocou algoritmu SOTree [6]

4 Algoritmus H3

Ako základ pre implementáciu nášeho algoritmu poslúžil algoritmus H3 publikovaný v roku 2000 v doktorskej práci Tamary Munzner s názvom *Interactive Visualization of Large Graphs and Networks* na Stanfordskej univerzite v Kalifornii [1].

Algoritmus H3 bol priamo navrhnutý na zobrazovanie štruktúry webových stránok a odkazov medzi nimi. Pre zobrazenie používa 3D hyperbolický priestor, ktorý je premietaný do euklidovskej gule v troj rozmernom priestore, ktorá je vykresľovaná na dvoj rozmerný monitor počítača. Jeho výhodou je starostlivo vybraná štruktúra kostry grafu a možnosť zmeny analyzovaného miesta v grafe pomocou techniky Focus+Context View – veľké okolie je viditeľné okolo práve analyzovaného miesta v grafe. Tým je dosiahnuté to, že aj pri veľkých grafových štruktúrach je v každom mieste grafu rovnaká informačná hustota.

Je uspôsobený na vizualizáciu špeciálneho typu grafov - kvazi-hierarchických grafov. Jedná sa o grafy, pre ktoré je pri stavbe kostry jednoducho definovateľný preferovaný predok uzlu zo všetkých odkazov do uzlu. Z pohľadu teórie grafov ide o graf s minimálnou kostrou, kde váhou na hranách je informácia o polohe uzla (adrese stránky). Jedná sa o orientované stromy, ktoré ale nemusia byť nutne acyklické. Výhodou však je, ak existuje čo najmenšie množstvo spätných hrán.

Kostra grafu je súvislý a acyklický graf, ktorý obsahu všetky vrcholy grafu ale nemusí obsahovať všetky hrany. Kostra (kostry) grafu môže byť vypočítaná pre ľubovoľný graf (les). Medzi štandardné algoritmy patrí Kruskalov alebo Primov algoritmus. V našom prípade pri tvorbe kostry grafu používame doplňujúcu informáciu, ktorou je URL (v praxi WWW adresa) indexovanej stránky.

Vizualizácia dát previazania webových stránok pomocou grafu je obzvlášť vhodná, keďže priamo

štruktúra DNS záznamov má stromovú štruktúru, a aj väčšina stránok má ako základnú organizačnú štruktúru strom. Tiež je veľmi dôležité, že zobrazovanie informácii pomocou stromového zobrazenia je pre ľudí prirodzené, intuitívne a veľmi jednoduché na pochopenie a ovládanie.

Hyperbolický priestor bol zvolený pre zobrazovanie preto, lebo poskytuje projekciu nekonečného priestoru do konečného vykresľovacieho priestoru a vďaka tejto vlastnosti môže mať každý uzol pridelený dostatočne veľký zobrazovací priestor tak, aby nedochádzalo ku kolíziám.

Prehusteniu, a tým pádom strate prehľadnosti, pri projekcii veľkých a hustých grafov sa predchádza pomocou možnosti prechádzania po grafe – približovaniu a odďaľovaniu jednotlivých uzlov a taktiež orezávaniu grafu podľa potreby.

V euklidovskom priestore môžu byť stromy vykreslené iba v prípade, že smerom k listom sa priestor pridelený jednotlivým uzlom exponenciálne zmenšuje, keďže počet uzlov sa vo vyváženom grafe exponenciálne zvyšuje. Potom v prípade analyzovania dát pri koreni stromu sa úplne stráca schopnosť simultálnej analýzy dát pri listoch z dôvodu exponenciálne odlišnej mierky.

Narozdiel od toho, môže v hyperbolickom priestore zaberať každý uzol približne rovnaký priestor, a preto každá časť grafu nesie rovnakú informačnú hodnotu.

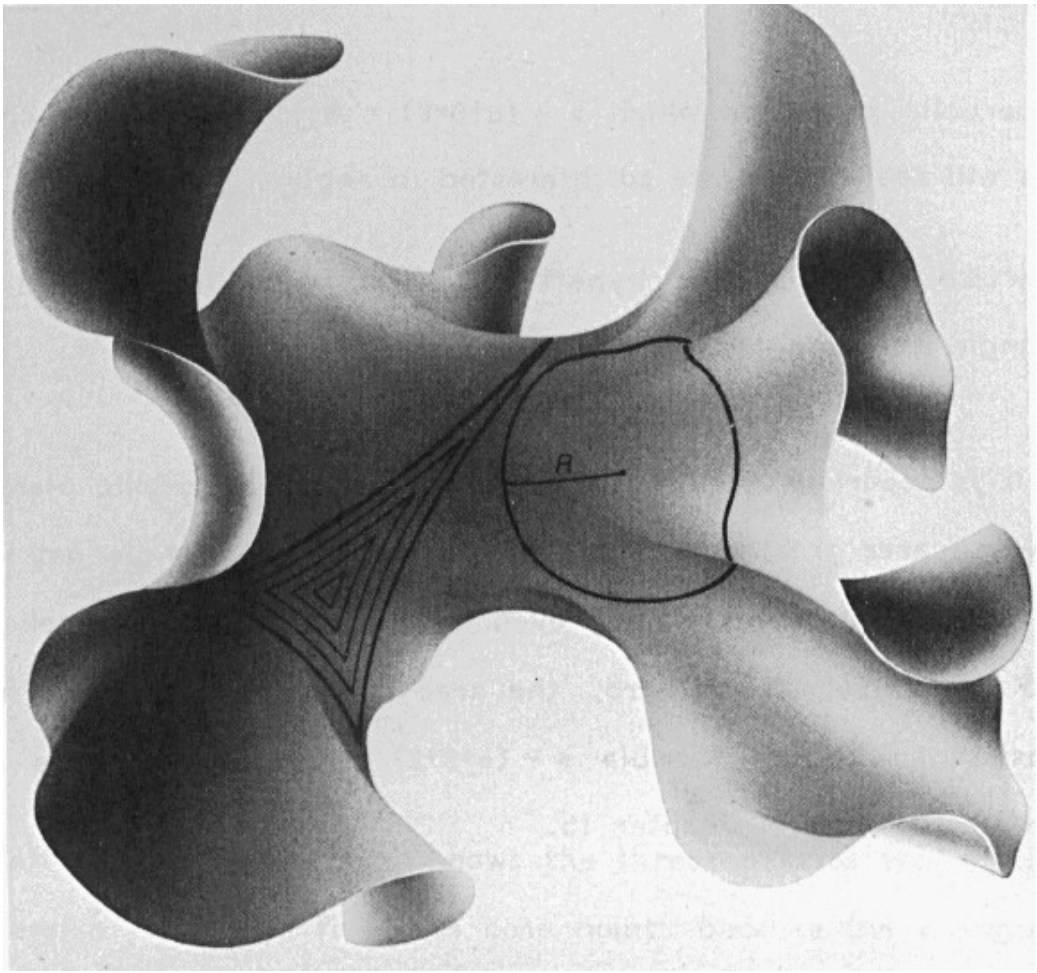
4.1 Hyperbolická geometria

Hyperbolická geometria je jedna z neeuklidovských geometrii vyvinutých na prelome 19. a 20. storočia. Má dve zásadné výhody pre zobrazovanie veľkých grafových štruktúr. Prvou je elegantný spôsob vykresľovania technológiou Focus+Context pomocou známej projekcie, ktorá mapuje celý nekonečný priestor na konečný vykresľovací priestor. Druhou výhodou je možnosť alokácie rovnakého priestoru pre každý z vrcholov v strome a stále sa vyhnúť kolíziám.

Hyperbolická a sférická geometria sú jediné dve neuklidovské geometrie, ktoré sú homogénne a majú izotropnú mieru vzdialenosti. Inak povedané, existuje jednoznačný a kontinuálny koncept vzdialenosti medzi dvoma bodmi. V prípade sférickej geometrie neexistujú paralelné čiary, v hyperbolickej geometrii je mnoho čiar prechádzajúcich jedným bodom a navzájom paralelných.

V hyperbolickej geometrii rastie plocha exponenciálne s polomerom, dôsledkom čoho je, že v hyperbolickom priestore je viac miesta ako v euklidovskom priestore.

Obvod kruhu v hyperbolickom priestore rasie exponenciálne podľa vzorca $2\pi \sinh r$, kdežto v euklidovskej geometri to je $2\pi r$. Na Obrázku č.7 je príklad 2D hyperbolickej roviny vloženej do 3D euklidovského priestoru, ktorý dáva predstavu o koľko viac priestoru je v hyperbolickej rovine oproti euklidovskej rovine.

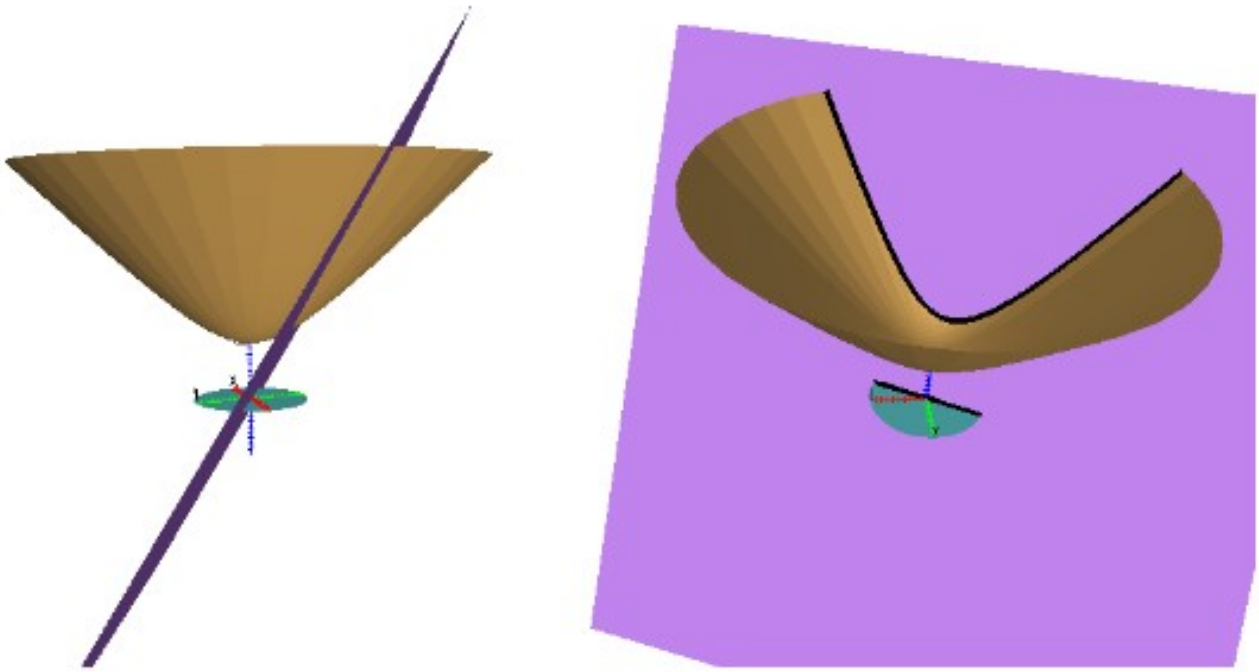


Obrázok 7: 2D hyperbolická rovina vložená do 3D euklidovského priestoru.

4.2 Projekčný model

Hyperbolický priestor (rovnako ako euklidovský) je nekonečný, ale existuje mapovanie na konečnú oblasť euklidovského priestoru. Odpovedá to euklidovskej kamere, ktorá by snímala obrázok celého hyperbolického priestoru *zvonka* poskytujúc Focus+Context pohľad.

Na Obrázku č.8 sú dva príklady príklad v 2D. Povrch hyperboloidu je premietaný na konečný euklidovský kruh. Príklad v 3D je náročné zobraziť formou obrázku, keďže by sa jednalo o 3D hyperboloid premietaný na guľu.



Obrázok 8: Príklad 2D hyperbolickej projekcie hyperboloidu do euklidovského kruhu. Na ľavom obrázku je rezová rovina kolmá na uhol pohľadu a na pravom je skoro paralelná

Exponenciálne narastanie priestoru s polomerom je veľmi vhodné pre zobrazovanie stromov, keďže aj u nich narastá počet vrcholov smerom k listom exponenciálne. Prvý krát bola idea takéhoto vykresľovania navrhnutá už v roku 1980 v práci pána Thurstona.

4.3 Rozloženie grafu

H3 algoritmus je rozšírením systému Cone Tree, ktorý vyvinula firma Xerox PARC. Tento algoritmus vykresľuje uzly do 3D euklidovského priestoru tak, že potomkovia vrcholu sú na obvode kužeľa vychádzajúceho z predka. V projekte WebViz, na ktorom sa spolupodielala aj Tamara Munzner, bola prvý krát využitá myšlienka z Cone Tree algoritmu v 3D hyperbolickom priestore.

Narozdiel od tohoto algoritmu sa v H3 algoritme vykresľujú potomkovia vrcholu na pologuľu na konci kužeľa, prípomínajúc kopček zmrzliny na kornúte.

4.4 H3 Viewer

H3 Viewer je prvotná implementácia algoritmu H3 (tzv. Proof-of-concept). Jeho upravená verzia bola použitá v komerčnom produkte Site Manager firmy SGI, a preto jeho implementácia nie je voľne dostupná.

Vstupom pre H3 Viewer je textový súbor, v ktorom každý riadok reprezentuje jeden vrchol grafu a má formát:

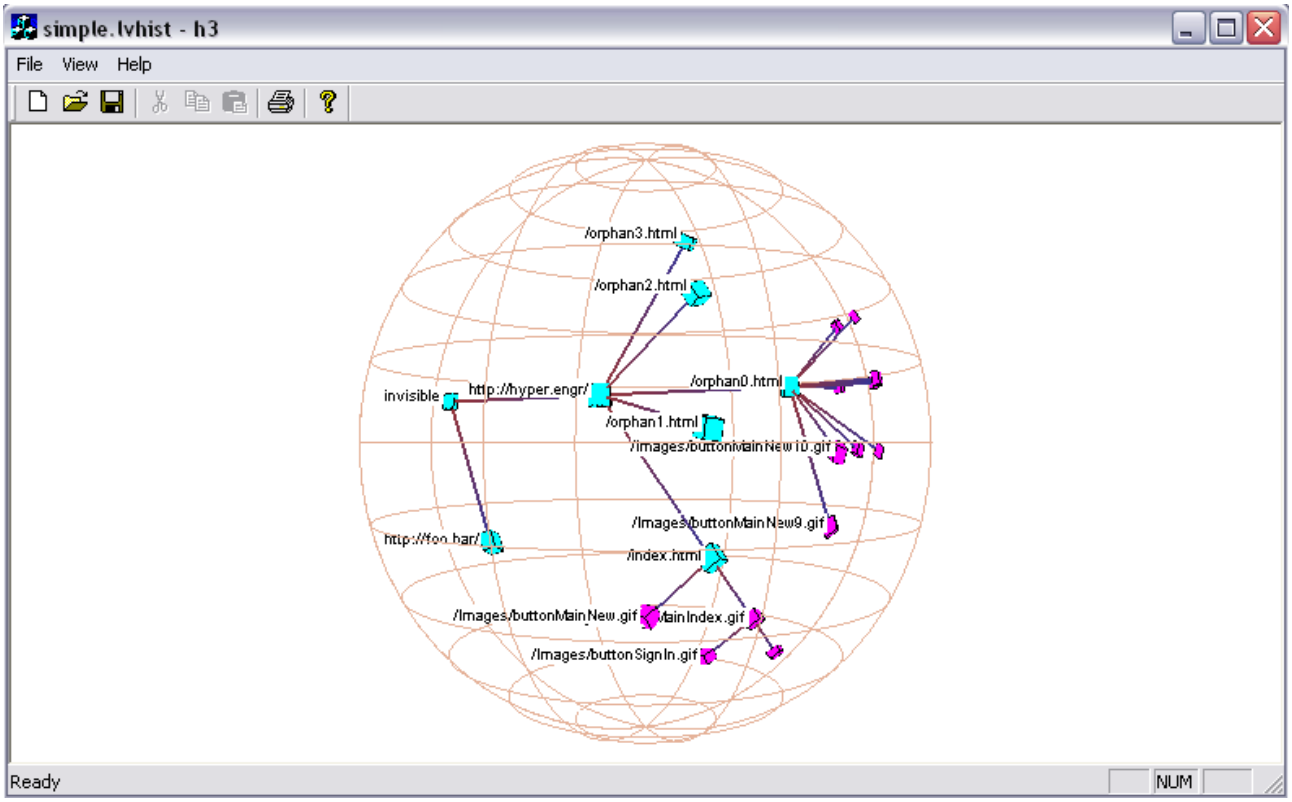
```
depth identifier 1 group1 [group2 ... groupN]
```

Príklad:

```
0 http://hyper/ 1 html main
1 http://hyper/index.html html main
1 http://hyper/logo.gif image main
1 http://hyper/old.html html orphan
```

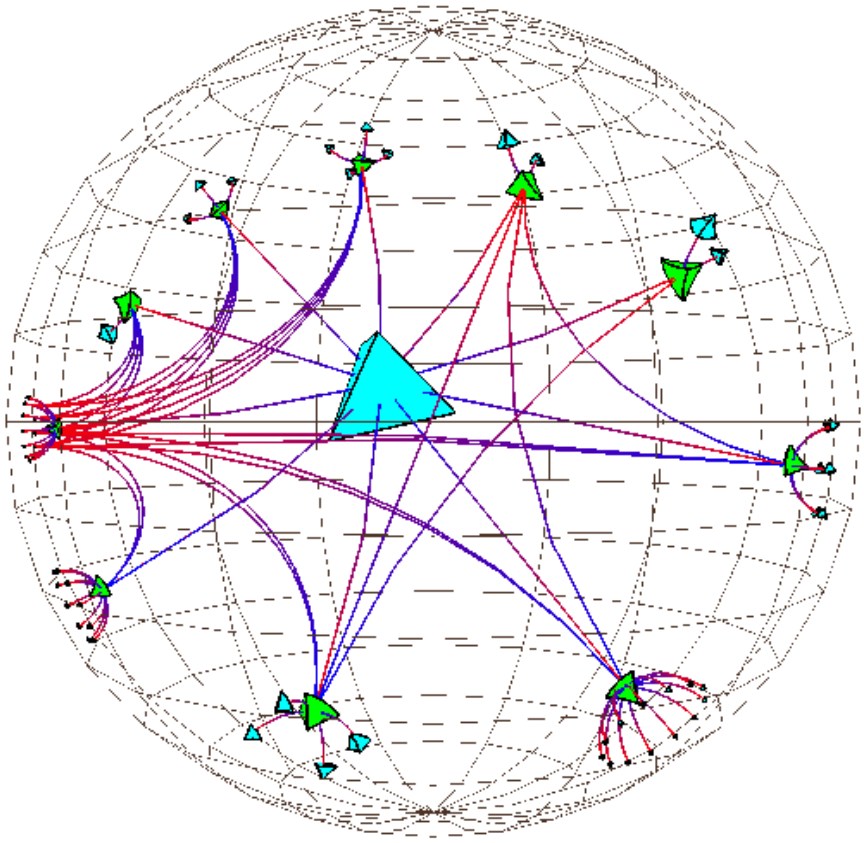
Výstupom je vizuálna reprezentácia grafu v hyperbolickom priestore premietnutom do gule v Euklidovskom priestore. Pri jednotlivých uzloch sú zobrazené URL adresy uzlov.

Zobrazenie jednoduchého grafu s cca 25 uzlami je na Obrázku č.9.



Obrázok 9: H3 Viewer s jednoduchým grafom

Predchodcom programu H3 Viewer bol projekt Webviz[4], pomocou ktorého bol analyzovaný princíp zobrazovania dát v hyperbolickom priestore a možnosti premietania grafu z euklidovského do hyperbolického priestoru (Obrázok č.10).



Obrázok 10: Webviz – conformal model projekcia

4 http://graphics.stanford.edu/papers/webviz/

5 WebView

Našim cieľom bola implementácia algoritmu H3 v jazyku Java na platforme JavaSE s použitím knižnice Java3D ako zobrazovacej technológie 3D grafiky. Vstupom (zobrazovanými dátami) boli indexové súbory vygenerované webovým robotom Egothor.

5.1 Egothor

Egothor je výkonný open-source textový vyhľadávací stroj, napísaný výlučne v Jave s podporou 64bitov, výkonným indexovacím algoritmom, inteligentným rozpoznávaním indexovaných dát a univerzálnym značkovacím algoritmom. Jeho implementácia prebiehala na Matematicko fyzikálnej fakulte Karlovej univerzity v Prahe pod vedením RNDr. Lea Galamboša a kolektívu prispievateľov.

5.2 Použité technológie

Platforma Java bola zvolená z dôvodu svojej multiplatformnosti a hlavne preto, že celý webový robot Egothor je naprogramovaný v Jave. Konkrétne bola zvolená desktopová verzia Javy – JavaSE – a to vo verzii 6, a preto je potrebné pred spustením aplikácie WebView nainštalovať na počítač JRE[5] tejto verzie. Veľkou výhodou tejto verzie Javy je prepracovaná podpora pre Swing (konkrétne Swing Application Framework) a celková integrácia s operačným systémom.

Z použitia Javy ako platformy vychádza aj použitie Java3D ako knižnice pri vykresľovanie akcelerovanej 3D grafiky. Táto knižnica poskytuje high level prístup ku grafickému procesoru

5 Java Runtime Environment (http://www.java.com/en/download/index.jsp)

a odtieňuje programátora od programovania konkrétnych grafických primitív, čo urýchľuje a zjednodušuje prácu s 3D grafikou a nevnucuje koncovému užívateľovi použitie konkrétnej implementačnej knižnice 3D rozhrania – podporované sú, okrem iných, aj najpoužívanejšie technológie na prácu s 3D grafikou - OpenGL a DirectX.

Nevýhodou použitia Javy3D bolo to, že jej základná komponenta pre vykresľovanie – *Canvas3D* – je tzv. heavyweight[6] komponenta, a preto jej súčasné použitie s lightweight komponentami Swingu môže pôsobiť problémy, a preto sme umiestnili vykresľovanie 3D grafiky do samostatného okna (*JFrame*). V novej verzii Java3D 1.5.2 už existuje aj lightweight komponenta *JCanvas3D*, ale prepis s použitím tejto komponenty nie je triviálnou operáciou, takže zostáva ako plán do budúcnosti.

Ďalšou použitou technológiou je ORM[7] mapovanie pomocou JPA[8] do databáze. Pomocou technológie JPA sa odtieňuje nutnosť používania SQL dotazov pri manipulácii s objektami v DB a taktiež nie je potrebné vytvárať pred použitím schému pomocou DDL príkazov.

Ako databáza je použitá Apache Derby[9] z dôvodu jednoduchosti použitia, nízkej spotreby pamäti RAM a možnosti použitia v embedded móde (nie je potrebná inštalácia serveru, databázový stroj je spustený v JVM klientského procesu).

Samostatná implementácia prebiehala na platforme Netbeans[10], a to konkrétne vo verzii 6.7+. Dôvodom pre výber tejto platformy bolo znova jej použitie pri vývoji aplikácie Egothor a hlavne jej excelentná podpora frameworku Swing a desktopovej Javy všeobecne.

6 Viac o heavyweight a lightweight komponentách na http://java.sun.com/products/jfc/tsc/articles/mixing/
7 Object-Relational Mapping (http://en.wikipedia.org/wiki/Object-relational_mapping)
8 Java Persistence API (http://en.wikipedia.org/wiki/Java_Persistence_API)
9 http://db.apache.org/derby/
10 http://www.netbeans.org

5.3 Inšpirácia

Ako inšpirácia pri našej implementácii H3 algoritmu poslúžil program H3Viewer[11], napísaný v jazyku *C++* s použitím knižníc *OpenGL/Mesa* a *STL*. a ďalšie práce Tamary Munzner na predchodcoch algoritmu H3, ktorých implementácie sú voľne prístupné – WebOOGL[12] a Geomview[13].

Ku programu H3 Viewer[14] sú voľne dostupné len zdrojové kódy rozhrania k samotnej vykresľovacej knižnici. Toto rozhranie je určené len na volanie binárnej vykresľovacej knižnice, ale aj tak poskytli určitý náhľad do dátovej štruktúry použitej pre H3 algoritmus a hlavne z nich vidieť algoritmy použité na stavbu a vykreslenie kvazi-hierarchického stromu v hyperbolickom priestore. Analýza týchto zdrojových kódov bola na začiatku našej práce hlavným zdrojom informácii a inšpirácie. Prepisom C++ kódu do Javy sme získali základ dátovej štruktúry a hlavne netriviálne matematické manipulácie s dátami potrebné pri výpočte a projekcii súradníc prvkov grafu medzi hyperbolickým a Euklidovským priestorom.

Implementácia algoritmu H3 existovala aj v komerčnom produkte Site Manager firmy SGI, ale tento produkt už dlho nie je ani podporovaný ani vyvíjaný. Program zobrazoval štruktúru konkrétneho webu a bol určený pre dizajnérov webových stránok.

Naša implementácia je v programe Java na platforme JavaSE s pooužitím technológie Java3D na vykresľovanie 3D grafiky, čím sa zabezpečí plná prenositeľnosť medzi rôznymi architektúrami.

11 http://graphics.stanford.edu/papers/h3draw/
12 http://www.geom.uiuc.edu/software/weboogl/
13 http://www.geomview.org/docs/
14 http://graphics.stanford.edu/~munzner/h3/

5.4 Tvorba kostry

Vytvorenie kostry vykresľovaných dát je prvým krokom pri ich spracovaní. V prípade vstupných dát vygenerovaných zo súborového systému hostiteľského operačného systému je kostra priamo daná, keďže väčeina súborových systémov má striktne stromový charakter.

Ak sú vstupným dátami zozbierané indexové súbory pomocou webového robota Egothor, tak na stavbu kostry grafu používame doplňujúcu informáciu a tou je adresa stránky. Začíname v koreňovom vrchole grafu a postupujem po hranách do šírky záplavovým spôsobom a podľa adresy www stránok sa vkladajú jednotlivé vrcholy do stromu.

Medzistavy tohoto prechodu sa ukladajú do databáze kvôli pamäťovej záťaži.

5.5 Vykresľovanie stromu

Vykresľovanie stromu prebieha v troch fázach. V prvej fáze sa zrátajú polomery pologulí pre všetky vrcholy od koreňa až po listy, v druhej fáze sa zisti uhly pre kužele na ktoré sa budú kresliť pologule s vrcholmi grafu a nakoniec pomocou týchto dvoch informácii sa vypočítajú súradnice jednotlivých vrcholov v hyperbolickom priestore, ktoré sú následne premietané do 3D euklidovského priestoru. Podrobnejší popis algoritmu a matematického pozadia sa nachádza v pôvodnej práci Tamary Munzner v kapitole *3.2.3 Tree Layout*.

5.6 Výber prvkov

Výber prvku grafu na užívateľov podnet (kliknutie myšou v scéne) prebieha na základe polohy reprezentovanej súradnicami výbraného bodu na scéne (x a y) a na základe 3D poradia (súradnice z) všetkých vrcholov, ktoré sa nachádzajú *v smere* výberu z pohľadu kamery. Nájdu sa

všetky vrcholy, ktoré majú určitú malú vzdialenosť od priamky pretínajúcej scénu v smere výberu a z týchto vrcholov je následne vybraný ten *najbližšie* k užívateľovi.

Taký to výber umožňuje dynamicky nastaviť toleranciu na výber podľa potreby tak, aby sa preferovali buď blízke vrcholy, ktoré ale nemusia byť presne v mieste výberu, alebo konzervatívnejši pohľad s výberom vrcholov veľmi blízko smeru výberu.

5.7 Dátové štruktúry

Základom dátovej štruktúry zobrazovaného grafu je myšlienka oddelenia zobrazovaných informácii a tzv. dodatočných dát, ktoré nie sú priamo nutné k vykresleniu grafu (popisné informácie vrcholov a hrán). Tým sa dosiahne to, že v pamäti (grafickej karty aj RAM) budú uložené len momentálne potrebné dáta a všetky ostatné informácie sa budú dohľadávať len pri určitej akcii užívateľa. Vzhľadom na vysokú náročnosť na CPU pri predspracúvaní zdrojových dát a na použitie pamäti RAM pri ich vykresľovaní je každá úspora v množstve spracovávaných dát dôležitá.

Preto pre reprezentáciu vrcholov, hrán grafu a ich vlastností (ako napríklad farba, viditeľnosť, súradnice v Euklidovskom aj hyperbolickom priestore) boli použité polia s presne definovaným poradím prvkov v nich. Kladie to určitú mieru zodpovednosti pri manipulácii s takto uloženými dátami, ale na druhej strane to prináša úsporu použitej pamäte oproti sofistikovanejšim objektovým modelom reprezentácie dát.

V najväčšej miere bol tento prístup použitý v triede *webview.graph.Graph*.

5.8 Vstupné dáta

Z licenčných dôvodov nie je možné pripojiť k tejto práci výstupné dáta z webového robota Egothor, a preto sú na priloženom CD k dispozicii testovacie dáta zobrazujce adresárovú štruktúru. Zobrazovací algoritmus pre oba typy dát funguje identicky a to tak, že dáta sú prekonvertované do interného formátu programu WebView.

Výstup webového robota Egothor je rozdelený do 4 typov súborov:

1. *pgw*.lnk* – popis štruktúry odkazov

```
1221 2: 3125 5554
4663 0:
1783 7: 0 5555 1000 5556 5557 5558 0
```

2. *pgw*.red* – popis štruktúry automatických presmerovaní (HTTP redirect)

```
19 121
78 606
5 831
665 1109
```

3. *pgw*.anc* – texty odkazov na stránkach

```
19 121
78 606
5 831
665 1109
```

4. *doc.aux* – názvy a adresy stránok

```
1 http://www.cuni.cz:80/
2 http://www.cuni.cz:80/UK-21.html
3 http://www.cuni.cz:80/?view=text
4 http://www.cuni.cz:80/UKENG-1.html
```

Podrobnejší popis sa nachádza na stránkach[15] projektu Egothor.

15 https://www.egothor.org/~galambos/twiki/bin/view/Egothor/RobotFiles

5.9 Štruktúra zdrojových kódov

Zdrojové kódy programu sú priložené na CD v podadresári *SOURCE* ako Netbeans projekt.

Obsah jednotlivých podadresárov adresára *src/* podľa abecedy:

- *META-INF/*
  súbor *persistence.xml* s nadefinovaným ORM mapovaním do embedded databáze Apache Derby[16], ktorá sa používa na ukladanie dodatočných informácii potrebných pri vykresľovaní grafu, tj. url stránok, texty a url odkazov zo stránok
- *webview/*
  okná a dialógy použité v programe
- *webview/base*
  JPA[17] triedy mapované do DB a obslužné triedy na načítanie grafovej štruktúry z DB
- *webview/generate/*
  testovacie triedy na generovanie grafových štruktúr
- *webview/graph/*
  triedy reprezentujúce vykresľovaný graf, základom je trieda *webview.graph.Graph*
- *webview/h3/*
  matematická, vykresľovacia a transformačná trieda pre hyperbolický priestor
- *webview/navigation/*
  navigačné triedy na obsluhu podnetov z polohovacieho zariadenia
- *webview/render/*
  triedy vykresľujúce graf (upravené Java3D vykresľovacie plátno Canvas3D, kresba os XYZ a trieda obsahujúca nastaviteľné parametre)
- *webview/resources/*
  doplnkové nastavenia pre Swing Application Framework

16 http://db.apache.org/derby/
17 http://en.wikipedia.org/wiki/Java_Persistence_API

5.10 Inštalácia

Ku spusteniu programu je potrebné mať nainštalované behové prostredie Javy vo verzii 6 alebo vyššej. Program bol testovaný len na referenčnej implementácii od firmy SUN[18] na operačných systémoch Windows XP a Linux (konkrétne Ubuntu 9.04).

Taktiež je potrebné mať nainštalovanú grafickú knižnicu Java3D vo verzii 1.5.2. Na operačnom systéme Windows XP stačí spustiť spustiteľnú inštaláciu zo stránok firmy SUN[19]. Na operačnom systéme Linux sme ešte museli nastaviť premennú JVM *java.library.path*, ktorá ukazuje na adresár s natívnymi knižnicami Javy3D.

5.11 Spustenie a ovládanie programu

Doporučené parametre pri spustení:

```
$> java -Xms256m -Xmx1536m -jar WebView.jar
```

Prvý parameter nastavuje počiatočnú veľkosť alokovanej pamäte pre JVM a druhý parameter nastavuje maximálnu veľkosť. Hodnota cca 1536MB je maximálna hodnota pre 32-bitovú JVM. V prípade, že sa na počítači nenachádzajú aspoň 2GB operačnej pamäti, tak môže byť táto hodnota znížená, ale treba počítať s degradovaním výkonu.

Voliteľný je parameter s cestou k Egothor súborom, ktorá bude prednastavená pre analýzu dát a druhý parameter určujúci, či sa majú vypisovať logovacie výpisy (vypnutím dôjde k zrýchleniu programu). K ideálnemu ovládaniu programu je vhodná kombinácia myši a klávesnice.

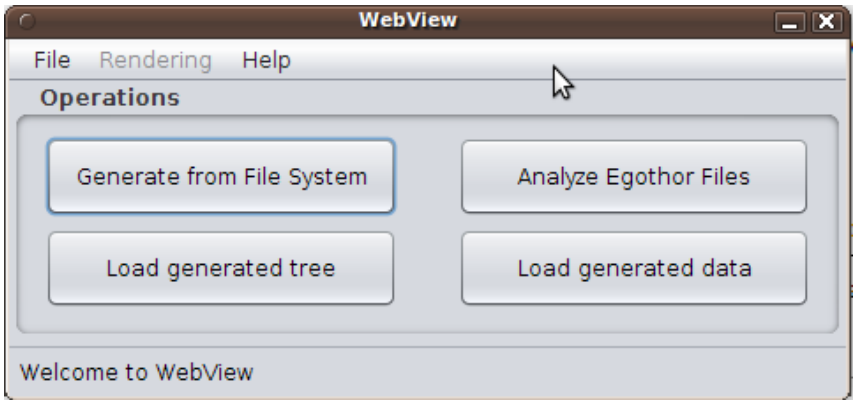
18 http://www.java.com/en/download/index.jsp
19 https://java3d.dev.java.net/binary-builds.html

V prípade, že nemáme k dispozícii vstupné dáta z robota Egothor, tak pre prezentáciu schopností H3 Algorithmu v programe WebView, môžeme použiť kvazi hierarchický graf vygenerovaný z adresárovej štruktúry súborového systému operačného systému.

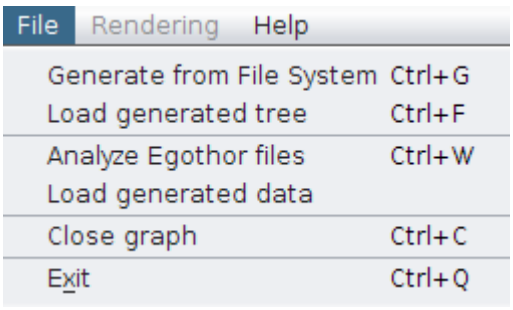
Pomocou položky menu alebo tlačítka *Generate from File System* sa vygeneruje strom z adresárovej štruktúry lokálneho disku zo zvoleného adresára a jeho podadresárov. Takto vygenerovaný strom sa potom zobrazí pomocou položky *Load generated tree*. V tomto prípade ale nie je možné zobrazovať informácie k vybraným uzlom, keďže nie sú dostupné. K dispozícii je plne funkčná navigácie po strome.

Štandardný postup spustenia začína vo vstupnej metóde *main* triedy *webview.WebViewApp*, ktorá zobrazí okno *webview.MainView*. V ňom pomocou menu alebo tlačítka vyvoláme analýzu externých vstupných dát z programu Egothor pomocou položky *Analyze Egothor Files*.

Ak už sme v minulosti nejaké dáta analyzovali, tak je prístupné aj položka *Load generated data*, ktorou sa zanalyzované dáta priamo vykreslia.

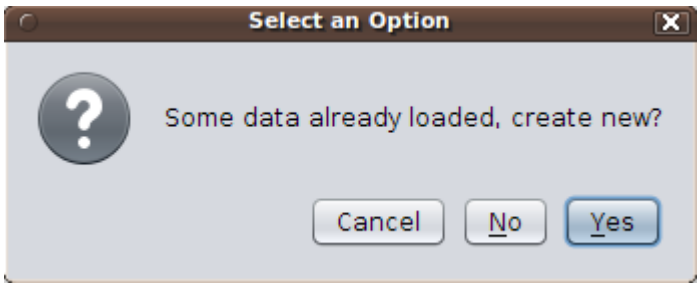


Obrázok 11: Hlavná obrazovka



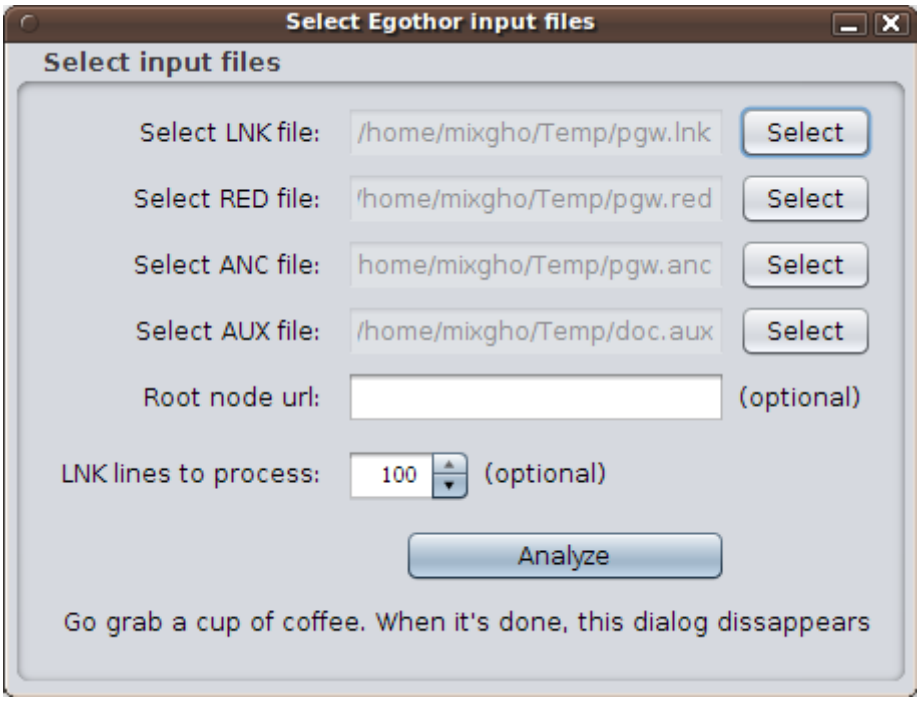
Obrázok 12: Menu File

V prípade, že už boli predtým zanalyzované, tak sa ukáže potvrdzovací formulár na premazanie dát (Obrázok č.13).



Obrázok 13: Potvrdenie premazania vstupných dát

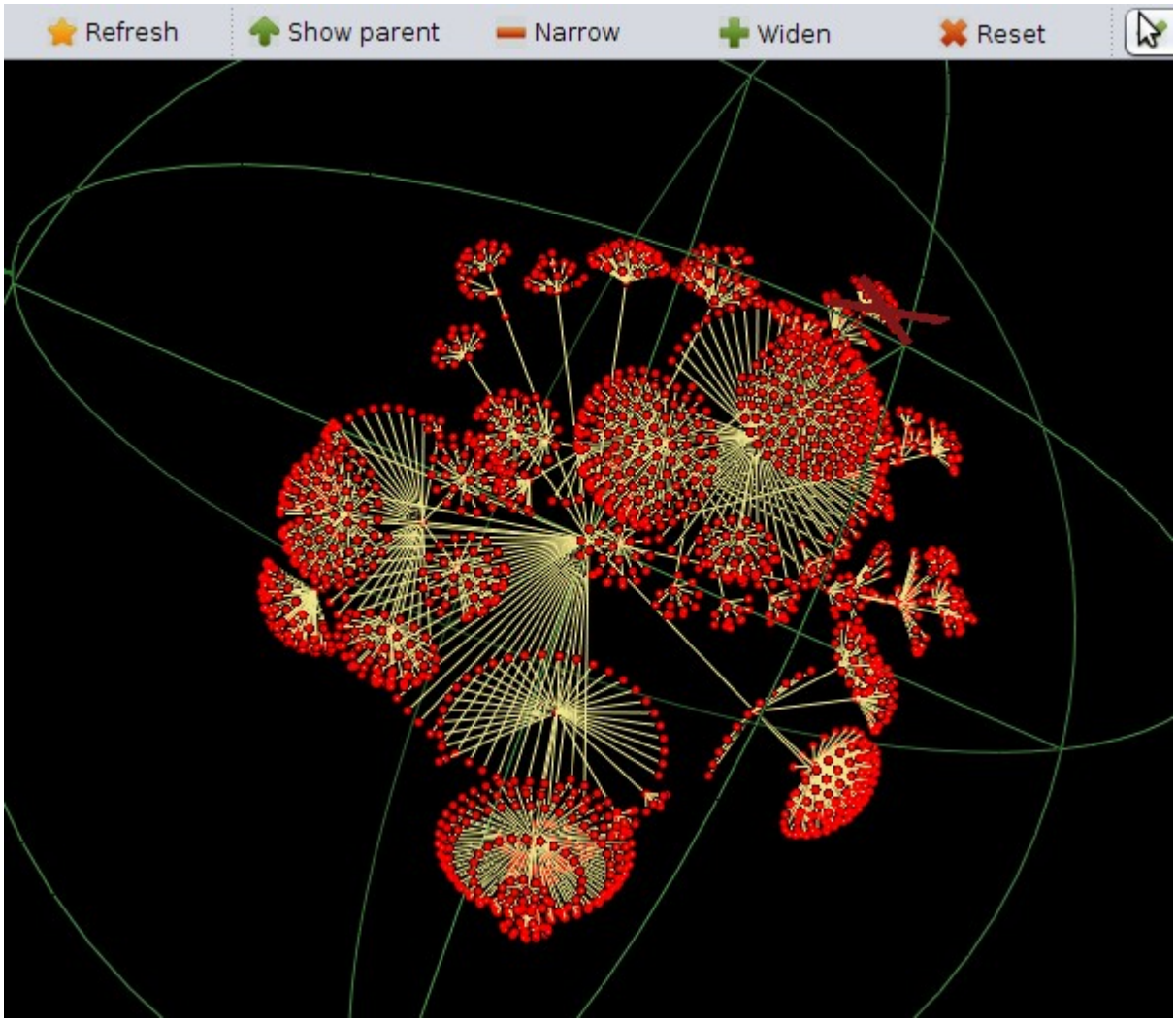
Po potvrdení sa zobrazí dialógové okno pre zadanie vstupných parametrov a vstupných súborov (Obrázok č.14).



Obrázok 14: Výber vstupných súborov z robota Egothor

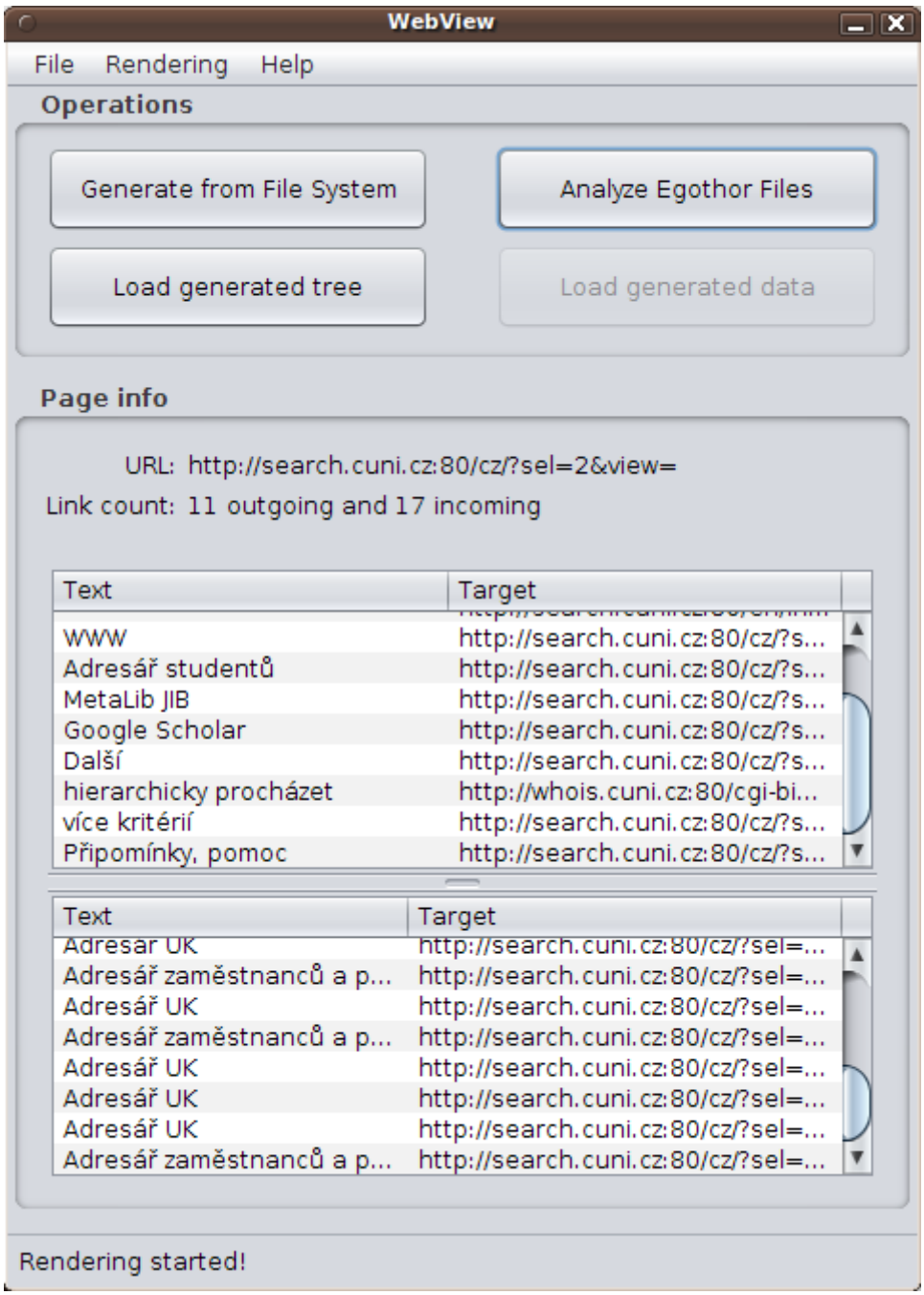
Po potvrdení tlačítkom *Analyze* sa spustí vstupná analýza a transformácia dát do dátovej štruktúry potrebnej na vykreslenie grafu a ich následné uloženie do DB. Táto operácia je časovo veľmi náročná v priamej úmere s veľkosťou vstupných dát od niekoľkých sekúnd po niekoľko hodín, preto je pre testovanie vhodné zadávať parameter *Graph size* dostatočne malý. Ak je špecifikovaný ako nula, tak sa spracujú všetky dostupné dáta.

Po skončení analýzi dát sa zobrazí samotný graf (Obrázok č.15).



Obrázok 15: Načítaný graf

V hlavnom okne (Obrázok č.16) sa zobrazujú informácie k vybranému uzlu (pomocou kliknutia myšou) a to jeho adresa a počet a popis jednotlivých odkazov z uzlu reprezentujúceho stránku (vstupných aj výstupných).



Obrázok 16: Informácie o vybranom uzle reprezentujúcom stránku

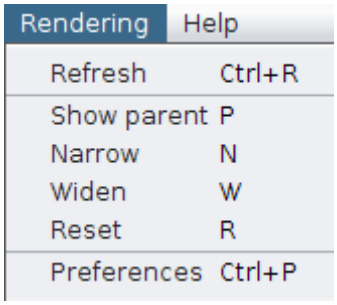
Navigácia grafu pomocou myši:

- *left-click* – výber uzla
- *right-click* – presunutie vybraného uzla do stredu vykresľovaného priestoru
- *drag-and-drop* – otáčanie zobrazovaného priestoru
- pohyb kolieskom myši – približovanie a odďaľovanie

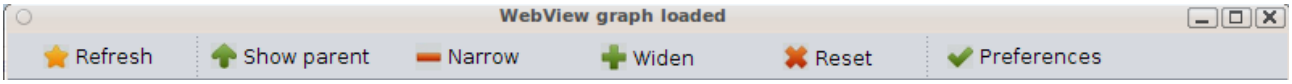
Navigácia pomocou klávesnice:

- N – orezanie grafu na podgraf vybraného uzla
- W – následne po orezaní grafu pomocou W nastane posun po ceste smerom do koreňa
- P – presun do predka po ceste do koreňa

Navigácie po grafe je tiež možná pomocou výberu položiek v menu *Rendering* (Obrázok č.17) alebo pomocou rýchlych volieb v lište nad zobrazeným grafom (Obrázok č.18).

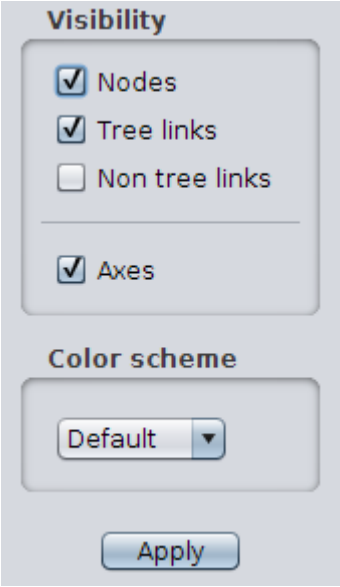


Obrázok 17: Menu Rendering



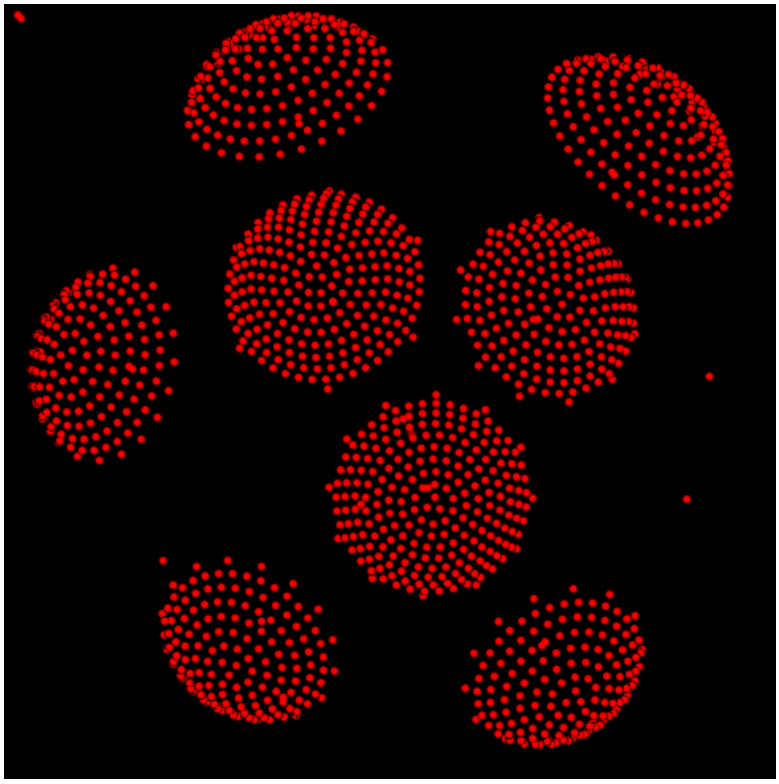
Obrázok 18: Operácie s grafom

Cez položku *Preferences* sa vyvolajú predvoľby nastavenia grafu (Obrázok č.19).



Obrázok 19: Nastavenia

Napríklad pomocou vypnutia (stromových aj nestromových) odkazov a os XYZ sa zobrazia len vrcholy grafu (Obrázok č.20).



Obrázok 20: Zobrazenie len vrcholov grafu

5.12 Dôležité triedy

- *webview.GlFrame extends JFrame*
  okno s Java3D obsahom
- *webview.base.Analyze*
  načíta vstupné súbory podľa zadaných parametrov v metóde a získanú dátovú štruktúry uloží do embedded databáze
- *webview.Graph*
  trieda reprezentujúca vykreslený graf, jeho vrcholy a hrany a ich vlastnosti
- *webview.graph.GraphLayout*
  najdôležitejšia trieda, čo sa týka vykresľovania, pomocou metód *computeRadii()*, *computeAngles()* a *computeCoordinates()* sa postupne rekurzívne prechádza graf a počítajú sa polomery, uhly a nakoniec pozície jednotlivých vrcholov grafu
- *webview.navigation.EventHandler implements KeyListener, MouseListener, MouseMotionListener, MouseWheelListener*
  obsluha vstupov (klávesnica a myš) od užívateľa
- *webview.render.MainRenderLoop implements Runnable*
  navigácia a manipulácia s grafom
  implementuje interface *Runnable* aby nedochádzalo k oneskoreniu alebo oneskorenému vykonávaniu príkazov od užívateľa
  dôležité metódy:
  - *render()* nastavuje, ktoré uzly a hrany sa majú vykresliť
  - *pickNode()* na výber vrcholu
  - *rotateDisplay()* na rotáciu zobrazovanej gule
  - *translate()* použitá pri presune vrcholu do počiatku
- *webview.render.VertexRenderer*
  stará sa o vykresľovanie uzlov a hrán grafu zo štruktúry triedy *Graph*

- *webview.render.ViewParameters*
  zoznam všetkých empirických konštánt a vizuálnych reprezentácii

Všetky triedy v zdrojových kódoch programu obsahujú stručný popis funkcionality.

5.13 Výsledná implementácia

Výsledná implementácia algoritmu H3 má viacero vylepšení oproti pôvodnej referenčnej implementácii v program H3 Viewer.

Vstupné dáta sú úplne oddelené od datových štruktúr potrebných pre vykresľovanie stromu, a preto sa na vstupe môžu používať rôzne dátové štruktúry. Zatiaľ sú implementované dva rôzne vstupné formáty a to vyexportovaná štruktúra súborového systému v operačnom systéme alebo výstupné dáta z webového robota Egothor, kedy je možné zobrazovať aj dodatočné informácie o pôvodných dátach (nie len štruktúru odkazov medzi jednotlivými stránkami ale aj ich názvy a presné URL).

Ďalším vylepšením je lepší vykresľovací algoritmus, ktorý umožňuje plynulejšie a intuitívnejšie presúvanie po veľkých a hustých grafoch. Bez väčších problémov (okrem úvodného času na zpracovanie dát) sme dokázali zobrazovať aj grafy s niekoľkými miliónmi prvkov.

Taktiež pomocou možnosti dynamických úprav farebných vlastností grafu, výberu množiny zobrazovaných vrcholov a ne/zobrazenia os priestoru je dosiahnuté prehľadnejšie využitie scény a takéto riešenie je aj príjemnejšie pre ľudské vnímanie vďaka podpore priehľadnosti a perspektívy.

Ďalšou veľkou výhodou oproti pôvodnej implementácii je možnosť orezávania grafovej štruktúry a vykresľovanie len určitých podstromov so samozrejmou možnosťou plynulého návratu smerom ku koreňu grafu. Táto vlastnosť spolu s technológiami Fan-out a Focus+Context umožňujú v reálnom čase prechádzať aj poveľkých grafoch bez straty celkového pohľadu až do namenších detailov.

Tiež sme sa snažili dosiahnuť pekný navrh zdrojových kódov programu s vysokym oddelením jednotlivých funkcionalít do samostatných tried s minimálnym previazaním. Takisto sme sústredili všetky empirické aj technické konštanty na jedno miesto, čo umožňuje jednoduchú zmenu chovania programu. Týmto sme dosiahli veľmi dobrú nahustenosť funkcionality vzhľadom na počet riadkov.

Nakoniec ale nie s najmenšou dôležitosťou sa jedná o otvorenú implementáciu nezaťaženú licenčnými podmienkami.

5.14 Problémy

Pri implementovaní programu WebView sme sa potýkali s viacerými problémami, ktoré sa nám buď podarilo vyriešiť (nedostatky vo frameworku Java3D) alebo sa jednalo o problémy všeobecného charakteru (nedostatok operačnej pamäte).

5.14.1 Matematický model

Samozrejme najväčšou problémom pri implementovaní algoritmu H3 bola zložitosť matematického modelu, na ktorom je postavený. Veľkou výhodou je, že k implementácii nie je potrebná znalosť matematiky hyperbolických priestorov, aj keď samozrejme určité porozumenie bolo potrebné. Nakoniec sa nám podarilo algoritmus správne naimplementovať tak, že program WebView spĺňa všetky predpoklady na neho kladené.

5.14.2 Dostupná pamäť

Najväčším problémom bolo obmedzenie veľkosti alokovateľnej RAM v 32-bitových operačných systémoch, ktoré nedovoľovalo prácu s veľkými grafmi s desiatkami miliónov vrcholov.

5.14.3 Nezrelosť frameworku Java3D

Ďalším, už spomínaným, problémom bola určitá nedokonalosť knižnice Java3D, keďže firma SUN upustila od vývoja tejto knižnice z dôvodu zamerania sa na iné zobrazovacie technológie – primárne JavaFX. To sa prejavovalo hlavne v nedokonalej spolupráci vykresľovacej komponenty Canvas3D s komponentami z knižnice Swing.

Veľkým nedostatkom bola aj nedostatočná dokumentácia – buď sa jednalo o veľmi zastaralú dokumentáciu k predchádzajúcim (už neaktuálnym) verziám Java3D alebo neexistovala dokumentácia vôbec. Toto je spôsobené nízkym počtom implementácii nad týmto frameworkom a utlmeným vývojom.

Pri vykresľovaní veľkých a hustých grafov (milióny prvkov) dochádzalo niekedy k pádu ovládača grafickej karty z dôvodu chyby v implementácii Java3D. V takom prípade bol potrebný reštart celého operačného systému.

```
[Java3D] Warning: Fail to lock Vertex Buffer - D3DERR_DRIVERINTERNALERROR
```

Naneštastie sa nám nepodarilo vypozorovať situáciu, kedy k tejto chybe dochádzalo.

5.14.4 Problémy s JVM

Pri práci s veľkými dátami generovanými z adresárovej štruktúry dochádzalo na platforme Linux k občasným pádom celého virtuálneho stroja JVM. Toto je spôsobené nie úplne ideálnou implementáciou I/O procedúr v balíčku *java.util*. V novej verzii Javy má byť práve z dôvodu veľkej nestability a nerobustnosti súčasnej implementácie nahradená novou.

6 Zhrnutie

Podarilo sa nám vytvoriť program na vizualizáciu dát generovaných pomocou webového robota Egothor. Program je schopný zobrazovať stromové grafy s niekoľkými miliónmi objektov v reálnom čase. Doba potrebná na ich predspracovanie zo surového vstupového stavu do stavu použiteľného pre WebView je síce značná, ale jednorázová, a preto akceptovateľná.

Škálovateľnosť programu je možná pomocou zväčšovania rozsahu pamäti RAM, keďže na 32-bitovom operačnom systéme s 32-bitovým JRE[20], ktorá má obmedzenie 2GB na jednu JVM[21] zvláda vykresliť do 50 miliónov objektov grafu, takže napríklad cca 10M uzlov a 35M hrán medzi nimi. V 64-bitovej Jave by sme sa mohli ľahko dostať k niekoľko násobným číslam. Vykresľovanie takto veľkých grafov je stále relatívne interaktívne.

Do budúcnosti sú možné tieto vylepšenia:

- postupné vykresľovanie grafu, ktoré by umožnilo zobrazovanie grafov, ktoré sa nezmestia do pamäte RAM
- rozšírené možnosti inteligentného zhlukovania uzlov a odkazov (pomocou dodatočnej informácie, napríklad pomocou rozkladu URL adries)
- vylepšenie vykresľovacieho algoritmu aby lepšie pokrýval celý zobraziteľný priestor
- úplné oddelenie vstupných dát od internej reprezentácie, čo by umožnilo pridávať podporu pre rôzne formáty dát vo forme zásuvných modulov

20 Java Runtime Environment - kolekcia behového prostredia a utilít
21 Java Virtual Machine – behové prostredie Javy

Prílohy

K diplomovej práci je priložené CD so zdrojovými kódami programu, s testovacími dátami a touto diplomovou prácou vo formáte PDF.

Bibliografia

[1] Interactive Visualization of large Graphs and Networks: Tamara Munzner 2000, http://graphics.stanford.edu/papers/munzner_thesis/all.onscreen.pdf

[2] Soumen Chakrabarti: Mining the Web: Discovering Knowledge from Hypertext Data. Amsterdam: Morgan Kaufmann, 2003.

[3] Hierarchy Browsers, Alexander Nussbaumer 2005, http://www.iicm.tu-graz.ac.at/thesis/anussbaumer.pdf

[4] Scalable, Robust Visualization of Very Large Trees: Dale Beermann , Tamara Munzner, Greg Humphreys 2003, http://www.cs.ubc.ca/~tmm/papers/contest03/

[5] EncCon: an approach to constructing interactive visualization of large hierarchical data : Quang Vinh Nguyen a Mao Lin Huang 2005, http://www.palgrave-journals.com/ivs/journal/v4/n1/full/9500087a.html

[6] Improvements of Space-Optimized Tree for Visualizing and Manipulating Very Large Hierarchies , Quang Vinh Nguyen a Mao Lin Huang 2007, http://www2.computer.org/portal/web/csdl/doi/10.1109/INFVIS.2002.1173152