VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D BROWSER FOTOGRAFIÍ

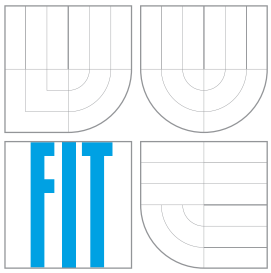
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZSOLT HORVÁTH

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D BROWSER FOTOGRAFIÍ

3D PHOTO BROWSER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ZSOLT HORVÁTH

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN

BRNO 2010

Abstrakt

Dnešní prohlížeče fotek již nejsou jen obyčejnými prohlížeči. Pravidelně se objevují inovace jako například sdílení fotek na internetu, sociálních sítích nebo jejich zobrazení v 3D. Naším cílem bylo vyzkoušet nové, ještě neexistující, 3D zobrazení. Porovnat ho s již existujícími a testovat jeho výhody i nevýhody. Byla vytvořena aplikace s podporou čtyř zobrazení, dvou již existujících a dvou zcela nových, s použitím OpenSceneGraph v jazyce C++. Nová řešení jsou založena na principu attraktory, tedy magnetů, které k sobě přitahují fotky podle zadaných parametrů. Aplikaci vyzkoušelo osm lidí. Každý dostal své úkoly, které musel splnit, a dotazník, který vyplnil. Z těchto úloh byly vyvozeny informace o použitelnosti, intuitivnosti a efektivitě. Výsledky ukazovaly, že existující řešení byla efektivnější, tato efektivita ale klesala s rostoucím počtem fotek. Při malém množství byla existující řešení dvakrát efektivnější.

Abstract

Today's photo browsers are no longer only browsers. Everyday are invented new innovations such as sharing photos on the Internet, on social networks or 3D vizualization. Our aim was to test a new 3D views which does not yet exists, to compare with the existing and test good and bad sides of designed solutions. Application was created with the support of the four views, two existing and two new, using the OpenSceneGraph in C++. The solution is based on the principle of attraktors, magnets, that attracts each photo according to specified parameters. Application tried eight people. Everyone got their tasks, which had to do and after that complete the questionnaire. These tasks have been derived information on usability, intuitive and efficiency. The result showed that the existing solution are more efficient, but efficiency decreased with more the number of photos. The existing solutions were twice more efficient with smaller number of photos.

Klíčová slova

3D browser, OpenSceneGraph, atraktory, OpenGL, knihovna, vizualizace, 3D fotky, kolekce

Keywords

3D browser, OpenSceneGraph, attractors, OpenGL, library, vizualization, 3D photos, collections

Citace

Zsolt Horváth: 3D browser fotografií, bakalářská práce, Brno, FIT VUT v Brně, 2010

3D browser fotografií

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana

.....
Zsolt Horváth
13. května 2010

Poděkování

Chtěl bych se poděkovat vedoucímu práce, Ing. Vítězslavovi Beranovi, za poskytnutí pomoci a podkladů a za podporu, kterou projevoval po celou dobu vytvoření této práce

Obsah

Seznam obrázků

Kapitola 1

Úvod

Programy, ktoré slúžia na prezerania fotografií nájdeme skoro v každom počítači. Klasické fotoaparáty boli vymenené na nové a moderné, ktoré sú už digitálne a umožňujú uložiť fotky do počítača. Ľudia začali ich skupinovať, ale po čase skoro u každého prevláda chaos. Na tieto účely boli vyrobené programy, ktoré miesto nás spravujú fotky a naviac umožnia nám nahrávať ich na internet a zdieľať ich s priateľmi. Jednoducho môžeme pridať komentáre či označiť, kto sa nachádza na konkrétnej fotke. Výrobcovia sa snažia lákať zákazníkov vytvorením krásneho grafického rozhrania a niektorí sa pokúšali experimentovať s 3D. 3D efekty používané v týchto programoch ešte zďaleka nie sú na takej úrovni, aby sme mohli rozprávať o ozajstných 3D prehliadačoch. Sú aj pokusy orientujúce sa na internet, všelijaké voľne dostupné flash galérie. Tieto sú len napodobeniny 3D, nemajú ani grafickú akceleráciu, je to simulácia trojdimenzionálneho priestoru bez podpory grafickej karty, napriek tomu fungujú dosť rýchlo a spoľahlivo.

1.1 Špecifikácie cieľov

Cieľom je vytvoriť knihovňu, ktorá umožňuje prezentovať inovatívne predstavy o zobrazení fotiek v trojrozmernom prostredí. Táto knihovňa bude obsahovať rôzne typy zobrazenia, medzi ktorými bude možné prepnúť a porovnať ich. Odskúšam jednoduchšie aj ťažšie typy. Porovnám už existujúce riešenie s novými, ktoré sú lepšie pre budúcich užívateľov. Slová jednoduchšie a ťažšie znamenajú, že sú typy, u ktorých užívateľ ľahko zistí, ako to ovládať, ale ja sa pokúsim vyskúšať niečo nového, čo ešte neexistovalo, zaviesť do programu `atraktory`. Pojem `atraktor` bude synonym na magnet, čo priťahuje určité fotky. To, ako ich bude priťahovať a presne ktoré fotky, to je otázka implentácie a predstavivosti. Knihovňa bude naväzovať na ďalšiu knihovňu, ktorá je nadstavba na `OpenGL`, `OpenSceneGraph`. Zdrojáky budú písané v `C++`, pri riešení sa pokúsim využívať návrhové vzory.

Kapitola 2

Rozbor

Táto kapitola je venovaná rozboru existujúcich riešení. Rozoberiem programy, ktoré sú na trhu, komerčné i nekomerčné, predovšetkým so zameraním na ich vlastnosti. Pozriem sa, čo všetko ponúkajú uživateľom: správa fotiek, užívateľké rozhranie. Budem skúmať ich 3D podporu a pokúšam sa inšpirovať už existujúcimi riešeniami. Rozoberiem knihovne `OpenSceneGraph` a `Qt`, ktoré slúžia na tvorbu 3D aplikácií a `GUI`.

2.1 Existujúce riešenie

Pictomio 3D

Na rozdiel od ostatných dvojdimenzionálnych režimov zobrazenia, režim *Carousel* predstavuje svoje snímky na trojdimenzionálnom pohľade s využitím vnútorného *Pictomio rendering engine*.

Vzhľadom k tomu, že technológia sa rozvíjala a stala sa dostupnou pre široké vrstvy užívateľov, takmer každý užívateľ začal nejakým spôsobom skupinovať fotky, či obrázky. Skutočnosťou však je, že nie sú profesionáli a zvyčajne v nich chýba schopnosť udržať si svoje veci správne organizované. Inými slovami, je väčšia pravdepodobnosť, že všetky pokusy končia v chaose, v náhodnom poradí, bez chronológie. Je jasné, že to znamená, že bude mať ťažkosti s ich efektívnou správou a ich prezentácia bude skôr menej atraktívna. Našťastie existuje nástroj na údržbu našich fotografií, ktorý sa nazýva *Pictomio*.

Grafické užívateľské rozhranie (`GUI`) robí dobrý dojem na užívateľov. Je to organizované do troch hlavných panelov, každý s odlišnou úlohu: ľavý panel pomôže preskúmať PC a vyhľadať priečinky s obrázkami, dolný ľavý panel umožní vytvoriť si zoznam obľúbených priečinkov a položiek, vo väčšej pravej časti sa zobrazí náhľad fotografií. Náhlad programu je na obrázku 2.1.

A teraz sa dostávame k najpríťažlivejšej prezentácii metód: *Carousel*. Môže sa použiť, keď sa chce urobiť dojem na publikum, alebo keď jednoducho sa chce zdôrazniť veľkosť fotografií. Určite sa bude páčiť každému, pretože takéto profesionálne prezentácie zvyčajne stoja veľa peňazí. Ďalší aspekt tiež stojí za zmienku: užívateľ má možnosť do svojej prezentácie zahrnúť nielen obrázky, ale aj videá a pesničky. Tento program umožnuje multimediálne zobrazenie.

V programe sú využité 3D efekty, napríklad takým spôsobom, že obrázky sú zobrazené v zásuvkách, ktoré sa tvoria ako by boli ozajstné 3D zásuvky. Do týchto zásuviek sú hodené fotky a môže sa hrabať medzi nimi s myšou. Obrázky sa môžu prehliadať v trojrozmernom

Obrázek 2.1: Pictomio 3D

prostredí tak, že aktuálny obrázok je v strede, ďalší a predchádzajúci sú bokom trošku zaklonené, a s prepínaním sa menia animovane.

Aplikácia je dostupná na adrese: http://www.pictomio.com/. Ďalsie informácie v [2].

Twins Visions

Málokto by si pomyslel, že môže byť program, ktorý pre používateľa ukáže niečo nového, ale Visions to dokázal. Tu je navyše úplne jedinečné trojrozmerné zobrazenie vďaka navigácie. Po inštalácii automaticky vyhľadáva obrázky (samozrejme iba vtedy, ak užívateľ chce), a ponúka aj správu pre on-line fotogalériu na Flickr. Ak sa zvolí táto opcia, potom pomocou programu môžu sa nahrať fotky na stránku, kde sa môžu zdielať.

V programe je najvýraznejšia a najzaujímavejšia časť vyššie uvedená trojdimenzionálna vizualizácia, to znamená, že obsah zložky ukladáva program na čierne listy a tým je vidno, ktoré obrázky obsahuje. Takto jednoducho sa dá prechádzať medzi obrázkami presne ako vo fotoalbume. Tieto karty majú možnosť konfigurovať, zmeniť polohu a dovoľujú zmenu situácie. Fotky z rôznych kariet pomocou drag-drop metódy môžu byť ľahko presunuté na požadované miesto. Náhlad programu je na obrázku 2.2.

Organizovanie svojho fotoalbumu je oveľa jednoduchšie v 3D formáte, môžu sa ľahko uložiť fotografie do rôznych albumov. Photo sharing umožňuje pre užívateľov uloženie údajov o svojich fotografiách a albume, ako napr: značky, termíny, komentáre, súhrny a poznámky. Obrázky sa môžu podľa potreby rozdeliť do albumov.

3D Showroom je hlavné miesto, kde sa môžu zobraziť a spravovať albumy. 3D Showroom umožňuje prechádzať zložky v 3D svete, spustiť slide-show z fotografií, spravovať albumy, vykonávať pokročilé vyhľadávanie pomocou dotazov v zbierke a i oveľa viac.

Aplikácia je dostupná na adrese: http://www.twins-solutions.com/. Viac o programe [4].

Flashloaded

Nie je to samostatný program, je to sada menších flash programov, ktoré sú schopné zobraziť fotky na internete formou 3D galérie. Umožňuje jednoducho ich zabudovať do internetovej stránky. Obrázky sú uložené v XML súbore, z ktorého program načíta cesty a nastavenie. Podporuje rôzne spôsoby zobrazenia, ako: *špirála*, *stena*, *guľa*, *kocka*, *cylinder*, *tunel*, atď.

Obrázek 2.2: Twins Visions

Užívateľ si môže vybrať, ktoré riešenie pre neho najviac vyhovuje. Samozrejme medzi uvedenými technikami sú aj také, ktoré nie sú najvhodnejšie na účely prechádzania obrázkov v 3D priestore, ťažko sa v nich orientuje, alebo ťažko sa v nich pracuje. Príčinou môže byť aj nevhodný model, či práca s myšou. Program zobrazuje fotky na interaktívnej 3D guli pomocou *Papervision3D* motora. Zahŕňa rôzne efekty otvorenia animácie plnej interaktivity, rovnako ako myš. Obrázok môžete priblížiť ku kamere po kliknutí. S tlačidlami na klávesnici umožňuje prechody medzi miniatúrami. Guľa reaguje na pohyb myši otáčaním okolo osi x a y, s možnosťou uzamknutia oboch osí. Priblížiť a vzdialiť z gule je možné rolovaním s myšou.

(a) kocka (b) špirála (c) guľa

Obrázek 2.3: Náhľady zobrazenia Flashloaded

3D špirála je prvok `Flash` galérie, ktorá zobrazuje obraz na interaktívnej 3D rotujúcej špirále, obrázok 2.3(b). Rôzne rozloženie dosahuje zmenou výšky a šírky špirály. Užívateľ môže ju posúvať; pohybovať hore / dole a priblížiť sa k špirále voľne.

Prvok *3D kocka* zobrazuje obraz na interaktívnom 3D rotujúcom štvorci, náhlad je na obrázku 2.3(a).

Rôzne vzhľady možno získať nastavením počtu riadkov na zobrazenie obrázkov. Kliknutím na miniatúru otvoríme veľký obrázok.

Užívateľ môže tiež prejsť, nakláňať a priblížiť štvorec. Vzhľad a správanie je flexibilné a prispôsobiteľné vďaka širokej škále nastavenia parametrov.

Obrázek 2.4: Komponenty Flashloaded

3D zásobník umožnuje zobraziť obrázky, ktoré sú jeden za druhým, táto technika je využívaná aj na platforme *Linux* s rôznymi 3D compozitormi na výber aktívnych okien (*coverswitch*). Ďalším variantom je, že obrázky sú poskladané na dve strany tiež pomocou 3D zásobníka a uprostred sa nachádza aktívny obrázok.

3D tunel je ďalší spôsob, čo ponúka *Flashloaded*. To je najmenej používateľný, nedá sa v ňom približovať, ani sa vzdialiť.

Obrázky sa zväčšujú kliknutím na nich, čo môže byť ťažké v prípade, keď sa vybral obrázok zo spodu. Nebude to určite rozšírený model v takej forme v budúcnosti.

3D stena môže byť plochá alebo sa dá nastaviť na akýkoľvek zakrivený tvar a vytvoriť skutočne unikátny vzhľad. Užívateľ môže rotovať, nakláňať a priblížiť steny. Stena mu umožnuje vytvoriť veľký počet variantov nastavením parametrov.

Aplikácia je dostupná na adrese: `http://www.flashloaded.com/`.

Cooliris

Cooliris urobí zo surfovania grafický zážitok. Berie snímky z navštevovaných webstránok, rozširuje ich na extra veľké veľkosti a potom umožňuje surfovať cez nich v rozhraní, ktoré sa javí, ako v najmodernejšom fantastickom filme [15].

V súčasnej dobe podporuje väčšinu z najpopulárnejších stránok, ako sú *Facebook*, *MySpace*, *YouTube*, *Picasa*, *Flickr* a pár ďalších. Tento plugin umožňuje prehliadať stránky na ktoré má podporu, na prerobenom trojrozmernom rozhraní, ako napríklad *deviantArt*. Môžu sa tam prelistovať obrázky pomocou *Coolirisu*. Otvorme `http://www.youtube.com` a zapnúť plugin. Po zapnutí hneď vidieť že *Cooliris* nám načíta len obrázky a videá z aktuálnej stránky. Kliknutím na obrázok sa nám zväčšuje a načíta originálnu veľkosť, v prípade, že je to video aj to spustí. Prelistovať obrázky je možné s posuvníkom, ktorý je umiestnený dole v okne, alebo myšou. Je to tak urobené, že pri ťahaní ide to do nekonečna, takže po čase začne zobrazovať fotky znova.

Rozhranie, ktoré je na vyhľadanie a uloženie záložiek, je dobre vymyslené, všetko je pod rukou, ale trojrozmerné rozhranie neprinieslo nič nového oproti existujúcim riešeniam. Fotky zobrazuje na 3D stene, obrázok 2.6, čo používajú aj iné firmy už dávnejšie. Možno je to tým, že tento typ je najprehľadnejší a najjednoduchšie ovládateľný. Môj názor po vyskúšaní je

Obrázek 2.5: Cooliris, webový prehliadač fotiek

taký, že je to len krásne a lákavé, ale určite by som ho nepoužíval na prehliadanie stránok, video, či obrázkov. Okrem toho, že nepoužíva akceleráciu grafickej karty, funguje dobre, aj animované efekty idú plynule. Bol stiahnutý už skoro 20 milionkrát, ale údaje o tom, že koľko ľudí ho používa, reálne neboli nájdené.

Obrázek 2.6: Zobrazenie na nekonečnej stene

Aplikácia je dostupná na adrese: http://www.cooliris.com/.

2.2 Prekonanie existujúcich riešení

Vytvoriť program, ktorý prináša vylepšenie a inovácie už existujúcich riešení nie je jednoduché. Vymyslieť si nové inovatívne veci, ktoré sú použiteľné robí ľuďom problémy, lebo musím sa zamerať na to, že vytvorím dielo nielen pre seba, ale aj pre iných užívateľov. Musím dávať pozor na to, aby používanie programu bolo čím viac intuitívne, aby nerobil prekážky pri používaní. Inak by užívateľia nevymieňali svôj obľúbený prehliadač za nový a pracoval som zbytočne.

Na internete je možné nájst veľa komerčných i nekomerčných programov, ktoré sú určené na prehliadanie obrázkov. Niektoré umožnujú užívatelom usporiadať fotky podľa určených kritérií, ako je napríklad dátum a miesto vytvorenia. Profesionálnejšie prehliadače ponúkajú užívateľom možnosť označenia ľudí na obrázkoch a zdielať ich na webe na sociálnych sieťoch. Takým spôsobom môžeme nájsť fotky o sebe u iných ľuďoch, s ktorými sme boli na spoločných akciách. Ďalším typom sú programy, ktoré venujú pozornosť trojdimenzionálnemu

priestroru a podporujú zobraziť fotky v 3D, kde sa dá vybrať z rôznych variantov rozloženia, ako je stena, kolotoč, kocka, špirála. Väčšinou sú používané len jednoduché primitívy (tvary), čo môže mať za dôvod to, že užívatelia sa zle orientujú v 3D.

Pri tvorbe môjho programu cieľom bolo prekonať už existujúce riešenie a vytvoriť inovatívne, čo sa mi viac menej podarilo. Či je to použiteľné, inovatívne alebo intuitívne, o tom sa musí rozhodnúť konkrétny užívateľ. Užívateľ si musí sám vyskúšať a ohodnotiť program, či ho považuje za použiteľný. Veľa ľudí stratí orientáciu v 3D, preto pre týchto nie je najvhodnejšie zobraziť fotky takým spôsobom. Podľa štatistiky sú to hlavne ženy.V dnešnej dobe už aj každodenné aplikácie používajú 3D zrýchlenie grafickej karty, začali zabudovávať 3D podporu užívateľského rozhrania,a to nielen prehliadače fotiek, ale aj prehliadače webových stránok. Existuje zásuvný modul aj pre prehliadače *Firefox* od firmy *Mozilla*, ktorá podporuje usporiadanie našich obľúbených stránok na 3D stene.

2.3 OSG (OpenSceneGraph)

`OpenSceneGraph` je hierarchicky usporiadaná stromová dátová štruktúra. Jej obsah je organizovaný pre výkonnejšie renderovanie. Vrchol stromovej štruktúry scene graph tvorí koreňový uzol (*root node*). Pod koreňovým uzlom sa nachádzajú uzly, ktoré organizujú geometriu (*geometry*) a vykreslovacie stavy (*statesets*) vzhľadu scény. Tieto skupinové uzly môžu mať nula či viac synovských uzlov.

`OpenSceneGraph` má rôzne typy uzlov s odlišnou funkcionalitou. Sem patria LOD (*level of detail*) uzly, transformačné uzly, ktoré umožnujú meniť polohu geometrií v scéne, switch uzly, ktoré sú schopné blokovať, či povoliť svoje synovské uzly. Objektovo orientovaný scene graph poskytuje širokú variabilitu pomocou dedičstva. Uzly majú spoločného rodiča a definujú svoju vlastnú špeciálnu funkcionalitu. Nízkoúrovňové renderovacie rozhrania nemajú podporu na priestorové organizácie a spôsoby uloženia dát od rôznych typov uzlov. `OpenGL` a `Direct3D` primárne sa zamierajú na možnosti, ktoré poskytuje grafický hardware [16].

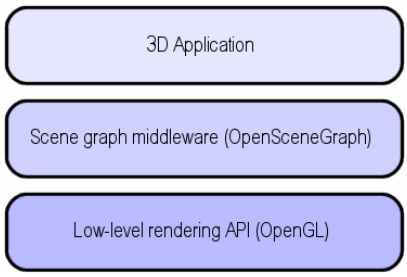
Hardware umožňuje uložiť stavy geometrií, ale nízkoúrovňové rozhrania nemajú možnosť na priestorové uloženie dát, čo nie je najlepšie pre väčšinu z 3D aplikácií. SceneGraph sa nachádza medzi nízkoúrovňovými rozhraniami a aplikáciami.

Nadštandartné vlastnosti scene graph:

- Priestorové organizácie: stromová štruktúra v scene graph.
- Odstrihnutie: zrýchlenie vykresľovania odstraňovaním neviditeľných častí.
- LOD (Level Of Detail): úroveň podrobností umožní efektívne renderovanie. Výpočíta vzdialenosti medzi objektom a kamerou, a časti scény načíta do pamäti, keď vzdialenosť od kamery je menšia, ako nejaká predom nastavená hodnota. Ak vzdialenosť je väčšia, objekt sa z pamäte automaticky odstráni.
- Minimalizácia zmeny stavu: pre maximálny výkon je dôležité sa vyhnúť zbytočnej zmene stavu. Scene graphy triedia geometrie podľa stavu pre minimalizáciu zmeny, odstráni redundancie.

- Knihovňa `OpenSceneGraph` poskytuje ešte mnoho funkcií okrem zmienených, ktoré nízkoúrovňové rozhrania nemajú, napr. podpora pre renderovanie textu, grafické efekty (particle effects, tieňovanie), optimalizácie pri renderovaní, podpora 3D modelov

`OpenSceneGraph` je *OpenSource*, cross-platformový grafický nástroj pre rozvoj vysoko výkonných grafických aplikácií, ako sú letecké simulátory, hry virtuálnej reality a vedecké vizualizácie. Je založený na koncepte `SceneGraph`, poskytujúci objektovo-orientovaný rámec na vrchole `OpenGL`. Je to rozsiahla knihovňa, ktorá podporuje objektovo-orientovaný prístup k návrhu 3D aplikácie. Bol navrhnutý ako platformovo nezávislý (*platform independent*). Napísané zdrojáky sú prenositeľné. Základom `OSG` je, že vytvorí orientovaný graf, kde uzly sú jednotlivé objekty. Takým spôsobom je možné budovať z jednoduchých objektov zložitejšie. Táto knihovňa je nadstavba nad `OpenGL`, ktorá je čiste objektovo-orientovaná. Všetky funkcie, ktoré existujú v `OpenGL` sú použiteľné. Existujú rôzne triedy svetla, materiálov. `OSG` je rozdelený na menšie balíky ako napr. `osg`, `osg::Util`, `osg:GA`, `osg:DB`. Tieto balíky obsahujú špecifické triedy, ako napr. `osg:GA` obsahuje abstraktné triedy na tvorbu GUI. Veľký nedostatok knihovne `OSG` je, že autori nevenovali veľa času na to, aby poriadne dokumentovali triedy a funkcie. Dokumentácia je dostupná, ale nie dostatočná na to, aby stačilo žaciatočníkom k vytvoreniu zložitejších programov. Sú dostupné príklady, ktoré pomáhajú k pochopeniu `OSG`. Najlepšie však je prečítať si knihu a tutoriály [6] a [11]. V `OSG` je protrebné vytvoriť nové objekty pomocou osg:ref_ptr, čo je ukazovateľ na objekt, a `OSG` automaticky uvoľní pamäť, keď na objekt už neukazuje nič. Je podobný, ako garbage collector.



Obrázek 2.7: OSG application stack z [12]

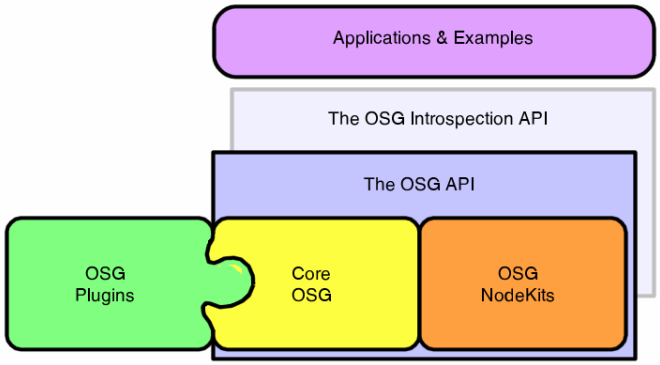
`OSG` je delené na menšie časti (obrázok 2.8):

- Jadro `OSG`: poskytuje základné modely a vykresľovacie metódy
- NodeKity: rozšírenie jadra `OSG` so špeciálnejšími uzlami a špeciálnymi efektmi.
- Zásuvné moduly: načítajú a zapisujú na disk 2D obrázky a 3D modely.
- Knihovne na podporu integrácie `OSG` do iných jazykov napr. do skriptovacích jazykov podobne ako k `OpenGL` (*PyOpenGL*) [5]
- Kolekcia aplikácií a príkladov na demonštráciu využitie knihovne `OSG`.

Jadro `OSG` obsahuje štyri hlavné časti: `osg`, `osgDB`, `osgUtil`, `osgViewer`. Časť `osg` je základom celého `OpenSceneGraphu`. Obsahuje uzlové triedy na modelovanie scény, ktorá vlastne je postavená z týchto uzlov. Obsahuje ďalej matice a vektory, ale nachádzajú sa tu aj geometrie modelov a implementované vykresľovacie stavy.

Všetky triedy uzlov sú odvodené z triedy `osg::Node`. Tieto uzly sa používajú pri stavbe scene graphu. Okrem základnej funkčnosti zdedeného od otcovskej triedy, každý uzol má vlastnú špeciálnu funkciu. Uzol *root* je vlastne `osg::Node`, ktorý nemá rodiča v scéne. Trieda `osg::Node` obsahuje metódy, ktoré pomáhajú vykresľovaniu, callback metódy a manažment stavu.

`osg::Group` je ďalšou dôležitou triedou, ktorá je základom všetkých uzlov, ktoré môžu mať potomkov. Je veľmi dôležitá z hľadiska priestorovej organizácie.



Obrázek 2.8: Architektúra OSG z [12]

Listovými uzlami grafu sú uzly typu `osg::Geode` (Geometry Node), môžu obsahovať objekty `osg::Drawable` obsahujúci geometrické modely na vykreslenie.

Uzol `osg::MatrixTransform` transformuje geometriu svojich potomkov podľa matice, ktorú obsahuje a rozhoduje o tom, či potomky budú otočené, posunuté, skosené. Vertex array z `OpenGL` predstavuje trieda `osg::PrimitiveSet` a `osg::Geometry`. Geometry obsahuje pole vertexov, súradnice textúry, pole normál a farieb. `OSG` umožňuje uložiť `OpenGL` stavy do scene grafu. Uzly sú triedené podľa stavov, redukuje ich, aby počet zmien stavov bol minimálny a renderovanie tým bolo rýchlejšie,. Ku každému uzlu môžeme priradiť nejaký stav, ale tieto stavy sa dedia podľa hierarchickej štruktúry pri vykresľovaní.

Reference-counted memory scheme je špeciálny pamäťový manažment v `OSG`, čo má za úlohu vyhnúť sa dieram v memórii. Bázová trieda všetkých uzlov a mnoho ďalších objektov je trieda Referenced. Obsahuje čítač odkazov na objekt. Objekty, ktoré sú odvodené od triedy Referenced a počet na nich ukazujúcich odkazov je nulový, bude zavolaný deštruktor. Podrobnejšie vysvetlené v [12], alebo v [17].

Hlavná sila v `QpenSceneGraph` je výkon, škálovateľnosť, prenosnosť a zvýšenie produktivity spojené s využitím plne vybavených grafických scenárov, podrobnejšie:

- **Výkon**
  Podporuje view-frustum culling, occlusion culling, small feature culling, Level Of Detail (LOD) nodes, `OpenGL` state sorting, vertex arrays, vertex buffer objects, `OpenGL Shader Language` a display lists ako súčasť jadra. Tieto spolu tvoria `OpenSceneGraph`, ktorá je jedna z najvyšších výkonných grafických nástrojov k dispozícii. `OpenSceneGraph` taktiež podporuje jednoduché prispôsobenie procesu kreslenia.

- **Produktivita**
  Jadro `OSG` vystihuje väčšinu funkcií `OpenGL`, vrátane najnovšieho rozšírenia, ponúka rendering optimalizácie ako utratenie a triedenie, a celú sadu add-on knižnice, ktoré umožňujú vytvoriť vysoko-výkonné grafické aplikácie veľmi rýchlo.

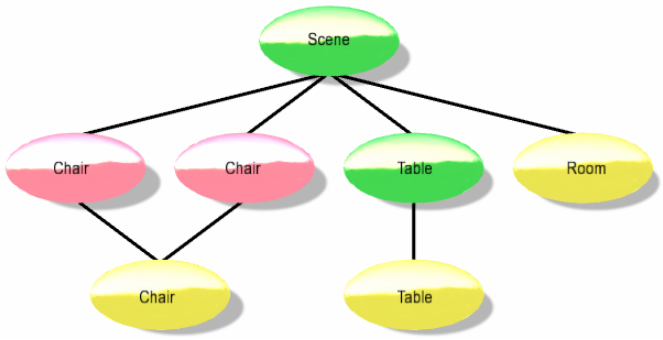
- **Prenosnosť**
  Jadro `OSG` bolo navrhnuté tak, aby bolo minimálne závislé od konkrétnej platformy, čo vyžaduje trochu viac, než štandardné `C++` a `OpenGL`, takto sa rýchlo rozšírili na celú radu platforiem - pôvodne vyvinuté pre IRIX, potom k Linux, Windows, FreeBSD, Mac OSX, Solaris, HP-UX, AIX, a dokonca aj PlayStation 2.

Jadro knižnice je úplne nezávislé na systéme, ktorý zobrazuje, čo umožňuje užívateľom jednoducho pridávať svoje vlastné špecifické knižnice. V distribúcii `OpenSceneGraph` `osgViewer` knihovňa poskytuje natívnu podporu okna pod Windows (Win32), Unix (X11) a OSX (Carbon).

- **Multijazyková podpora**
  Java, Lua a Pythonu pre `OpenSceneGraph` sú k dispozícii ako Community projects.

Správa 3D dát s hierarchickou štruktúrou

Knihovňa má zabudovanú správu pre dáta, ktoré uloží do hierarchickej stromovej štruktúry. Ako už bolo písané, základom celej scény je koreňový uzol. Na tento uzol sa napájajú ostatné a vytvoria sa tým zložitejšie objekty. Existujú špeciálne uzly, ktoré umožňujú skupinovať ostatné, tzv. skupinové uzly. Treba si predstaviť jednoduchý strom, ktorý má dve vetvy. Kde sa rozdelia vetvy, tam bude skupinový uzol, na ktorý budú napojené. Táto stromová štruktúra umožňuje definovať statesety tak, že keď sa to nastaví pre koreňový uzol, bude to platiť aj pre všetky synovské uzly, ak nie je explicitne zakázané. Nastaví sa koreňovému uzlu osvetlenie, tak aj synovské uzly budú osvetlené.



Obrázek 2.9: Správa dat v OSG z [12]

Aplikácia používa skupinové uzly na organizovanie a usporiadanie geometrií v scéne. Treba si predstaviť 3D databázu obsahujúcu izbu so stolom a dvomi rovnakými stoličkami. Môže sa zorganizovať pre scénu táto databáza rôznymi spôsobmi. Obrázok 2.9 ukazuje príklad organizácie. Koreň má štyri synovské skupinové uzly, jeden je pre geometriu izby, jeden pre stôl a jeden pre každú stoličku. Skupinové uzly stoličky sú farebne odlíšené červenou, indikujúce to, že transformujú svoje synovské uzly. Existuje iba jedna stolička listového uzla, pretože tieto dve stoličky sú identické, skupinový uzol pomocou transformácie vytvorí z nich dve na rôznych lokalitách. Skupinový uzol stôl má jeden synovský uzol, ten je už koncový. Listový uzol izby obsahuje geometrie podlahy, steny, strechy.

Typicky sú realizované tri prechody scény počas generovania jednotlivých rámcov. Sú situácie, keď je potrebné generovať súcasne viac pohľadov scény. Táto technika sa volá stereo rendering, na realizáciu je tu príklad. V týchto situáciach aktualizačný prechod scény prebieha len raz za snímok, ale orezávanie a vykresľovanie prebieha raz za každý snímok pre každý pohľad. Táto technika umožňuje systémom s viac procesormi a grafickými kartami spracovanie graf scény paralelne.

2.4 Qt (framework)

Qt je open-source knihovňa hlavne na tvorbu GUI. Je to nezávislá na platformu, napísané zdrojáky sú prenesiteľné aj na iné operačné systémy, a to nielen na PC, ale aj na mobilné telefóny. Dokumentácia je veľmi rozsiahla, aj kvalitne napísaná. Existujú voľne dostupné nástroje, v ktorých je možné jednoducho ladiť programy, či štýlom *drag n' drop* poskladať grafiké rozhrania. QtCreator obsahuje zabudovaný debugger, ktorý potrebuje nastaviť potrebné parametre pri kompilácii, aby bolo možné zapnúť ladenie programu. Qt od verzie 4 je používateľný aj na negrafické aplikácie, ako sú konzolové programy. Knihovňa je písaná v C++, ale existujú rozšírenia, ako napr. pre Python (*PyQt*). Signály a sloty slúžia na prepojenie súčiastok GUI. Ak nastane nejaká udalosť, je generovaný obslužný signál. Tieto signály sú schopné zachytiť objekty, ktoré majú pripojený slot na tento signál. Pripojiť ho je možné pomocou funkcie connect, ktorá je definovaná v QObject. To znamená, že nové triedy by mali dediť od tejto triedy, a použiť makro Q_OBJECT [3].

Knihovňa používa vlastný program na vyrobenie Makefile, ktorý dostal meno qmake. Závislosti qmake vyrieši za nás, stačí len nastaviť zdrojové a hlavičkové súbory u jednoduchších programov.

Vo vývoji softvérového sveta existuje veľa nástrojov a služieb dostupných na pomoc vývojárom maximalizovať použitie vývojových cyklov a využiť každý bit funkčnosti z ich produktov. Tieto nástroje zahŕňajú kompilátory, debuggery, vývojové prostredia, profiler a samozrejme grafický Toolkit Qt, ktorý je rýchlo rastúci vlajkový produkt od nórskej firmy Trolltech. Ich motto je "Code Less. Build More. Compile Anywhere."motto reality, čo znamená, "Kóduj Menej. Stavaj Viac. Prelož kdekoľvek.".

Qt je multiplatformový nástroj. Na prvý pohľad sa zdá, že neprináša nič nového, ako príklad proti Javy, ale Qt ponúka trochu iný spôsob vytvárania grafických aplikácií. Po prvé, Qt je natívny toolkit. To znamená, že ak sa vytvorí aplikácia Qt a preloží sa, je vytvorený natívny binárny program, ktorý beží na konkrétny operačný systém. Keď je potrebný aj na iný operačný systém, tak stačí preniesť zdrojové súbory a znovu ich preložiť. Koncepcia je, že zdrojový kód je rovnaký pre každú platformu. Rýchlosť aplikácií sa výrazne zvýši, a rozsiahle aplikácie by mali fungovať teoreticky tak rýchlo, ako akýkoľvek iný natívny softvér, s výhodami Qt cross-platform zdrojákov.

Prvá a najzákladnejšia sada funkcií je pre tvorbu grafického rozhrania. Qt obsahuje widgety na gombíky, zaškrtávacie políčka, prepínače, taby, ikony, plátna, dialógové okná a ďalšie bežné prvky používateľského rozhrania. Okrem týchto prvkov tam sú špeciálne dialógové okná (napr. výber súborov, about box, atď) a zariadenia, ktoré zachránia vývojárov od toho, aby museli implementovať tieto funkcie znova a znova. Okrem týchto grafických prvkov Qt obsahuje rad doplnkových funkcií. Qt poskytuje niektoré z týchto tried na podporu: pole, reťazce, vektory, mapy, a mnoho ďalších typov. Okrem týchto základných tried sú tu aj triedy pre prácu v sieti, sockety, prenos súborov, zvuk a mnoho ďalších pre vývoj software. Je dobré vidieť, že Trolltech vytvoril bohaté API, ktoré poskytuje nielen grafické widgety, ale tiež poskytuje množstvo tried, ktoré každodenné úlohy programovania robia jednoduchšie. Extra moduly majú špecifické funkcie, ktoré môžu byť pridané k API. Tieto moduly obsahujú grafické plátno, prístup do databázy, vytváranie sietí, OpenGL a XML zariadenia.

Qt toolkit vychádza z jazyka C++, bola navrhnutá objektovo orientovaným programova-

ním (OOP). Každý z komponentov je k dispozícii ako trieda (napr. `QPushButton` vytvorí tlačidlo), a každá trieda má niekoľko metód na zvládnutie bežných úloh a funkcií. Dedičstvo je základným predpokladom pre `GUI`. Ako príklad je všeobecné tlačidlo, potom sú špecifickejšie typy z tlačidiel (push, radio atď). V `Qt` je dedičnosť používaná týmto spôsobom. Tento proces dáva vývojárom všetky funkcie, ktoré by mohli byť potrebné pre daný konkrétny widget, a nižšiu úroveň funkčnosti pre jeho zdedené triedy. Je neuveriteľne flexibilný a je implementovaný aj v `Qt`.

Jednou z najzaujímavejších vlastností `Qt` je spôsob, akým sa zaobchádza interakciou užívateľa. Zachytenie udalosti je zodpovednosťou programátora, robiť niečo konštruktívneho v reakcii na to. Trolltech prijal rafinovaný koncept, riešenie s názvom *signály a sloty*. Základnou myšlienkou je, že každá trieda má niekoľko predefinovaných signálov, ktoré sú emitované vtedy, keď sa niečo stane, napríklad kliknutím na tlačidlo alebo výberom položky v menu.

Jeden z nástrojov `Qt` je `qmake`. Tento jednoduchý malý nástroj umožní spracovať aplikácie ľahko. Mnoho vývojárov rozdelí ich kód na viac zdrojových súborov, a tradične vývojári musia upraviť `Makefile` a ďalšie budovacie skripty, aby ich aplikácie bolo možné preložiť. Qmake sa snaží zjednodušiť proces, a pri spustení bude generovať `Makefile`, čím je možné preložiť aplikáciu.

Kapitola 3

Návrh

V tejto kapitole je popísaný návrh knihovne, ako som postupoval. Obsahuje techniky, ktoré boli používané pri tvorbe programu. Sem patria hlavne návrhové vzory a možnosti objektovo orientovaného programovania. Vysvetlím pojem `atraktorov`, na aké účely sú použité, v čom je ich sila a príklady na pochopenie použitia. Rozoberiem vlastnosti používaných knihovien `QpenSceneGraph` a `Qt`. Preštudujem prostriedky pre správu 3D dát v `QpenSceneGraph`. Pozriem sa na schematický popis navrhnutých zobrazení.

3.1 Postup riešenia

Pri návrhu riešenia bolo potrebné pozbierať už existujúce programy, ktoré používajú 3D vizualizáciu. Tieto programy sa väčšinou podobajú, čo sa týka riešenia vizualizácie. Je u nich možné zvoliť aj z viacerých typov zobrazenia, ale žiadny z nich neponúka užívateľom nijaké nadštandartné vylepšenie oproti ostatným, alebo podporu rozloženia v 3D priestore. Užívateľom nie je možné dynamicky meniť a ovládať scény. Scény sa chovajú staticky, nie je možné premiestniť fotky. Väčšina týchto prehliadačov okrem efektov neposkytne nič podstatného pre užívateľov. Najpopulárnešími typmi vizualizácií sú *kolotoč* a *3D stena*. Realizovanie 3D priestorov u takých programov nie je jednoduché, návrhári musia dávať veľký pozor na to, aby scény boli prehľadné a jednoducho ovládatelné. Mnohým užívateľom robia tažkosti orientácie v trojdimenzionálnych priestoroch vo virtuálnej realite. Luďský faktor je u takých návrhov veľmi dôležitá vec, preto je potrebné vždy získať spätnú väzbu, analyzovať výsledky a prispôsobiť program k požiadavkám, aby sa nestalo, že výsledný program bude prekázať v práci, namiesto toho, aby ju uľahčila.

Mojím cielom pri tvorbe tohto demonštračného programu je vyskúšať si niečo nového a porovnať už existujúce návrhy s novým návrhom.

Vytvorím menšie balíky na špeciálnejšie úlohy. Tieto balíky budú obsahovať triedy jednotlivých zobrazení, jadro programu, alebo triedy, ktoré budú zodpovedné za dynamiku obrázkov. Ku každému obrázku je možné pridávať dynamické elementy, ktoré sa dajú zvoliť podľa potreby. Mám prichystaný balík dynamických efektov a stačí mi len vybrať a používať ich. V týchto triedach budú definované matematické operácie, výpočet pozícií, posunutie na požadované miesto, priblíženie, atď. Tieto matematické operácie musia byť vždy efektívne a rýchle vypočítateľné, aby sa nezdržovalo generovanie snímkov. Najlepšie je využívať podporu grafickej karty na matematické výpočty, lebo grafická karta má v sebe zabudované prostriedky na zložité operácie, čo je oveľa rýchlejšie, ako výpočet v centrálnej jednotke.

Sada obrázkov bude popísaná v súboru CSV, ktorý bude obsahovať cestu k obrázkom a

nastavené parametre, tzv. váhy k jednotlivým skupinám, triedám. Tento súbor mi zabezpečí tretia strana. Program získava informácie o fotkách z týchto súborov. Tieto súbory budem používať u zobrazeniach s `atraktormi` (magnetmi), u iných stačí zvoliť len adresár s fotkami a program mi automaticky natiahne z neho všetky informácie.

Mám hotový program, musím overiť, či som vytvoril použiteľný nástroj. Na overenie vymyslím úlohu, ktorú dostanú užívatelia s testovacími účelmi. Úlohy skúmajú intuitívnosť a efektivitu môjho programu. Porovnávam rôzne zobrazenia, ktoré boli implementované. K tomu je venovaná kapitola 5.

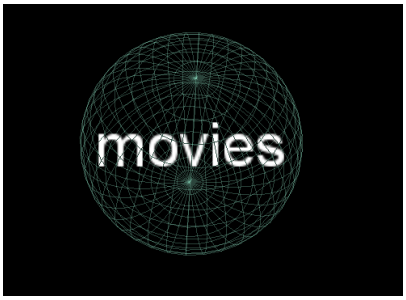
3.2 Návrh realizácie prehliadača

Z hľadiska realizácie musím rozobrať celý systém na menšie časti, moduly. Tieto moduly budú realizovať vykonávanie špeciálnejších úloh. Podobé moduly, napríklad, ktoré sa starajú o dynamiku, alebo tvoria jadro systému budú umiestnené do spoločného balíka. Štruktúra celého systému je znázornená na obrázku 4.1. Ako vidíme, systém je rozdelený na viac častí. Základ systému tvorí balík *engine*, ten obsahuje najpodstatnejšie funkcie, na neho naväzujú ostatné balíky. Balík *view* je napojiteľný na *engine*, jeho obsah je pre užívateľov najdôležitejší, totiž tam sú umiestnené jednotlivé zobrazenia, kolotoč, stena, kolotoč s `atraktormi`, atď. Do týchto zobrazení sú pridané špeciálne uzly vytvorené podľa knihovňe `OSG`. Uzly vedia spracovať obrázky, vytvoriť z nich menšie, načítať rôzne formáty a veľkosti, otáčať a zobraziť ich. Na tento balík naväzuje ďalší, ktorý je správcom dynamiky v našom systéme. Všetky operácie súvisiace s animáciou, premiestnením, výpočtom nových pozícií patria sem. *Collection* je len menšia časť, obsah tvoria hlavne prepravky, nosiče informácií, podrobnejšie v [14] a analyzátor súborov s dátami o fotkách. O interakcie s užívateľom sa starajú balíky *manipulator* – ovládače scény, iný pre každý typ zobrazenia, a *handler* – kliknutie a premiestnenie v scéne. Na koniec potrebujem ešte prvky grafického užívateľského rozhrania, tie budú v *gui*, zabalia môj systém do okna a pridajú panely, gombíky, atď.

3.3 Atraktory

`Atraktor` sa môže predstaviť ako magnet, ktorý určitým spôsobom priťahuje veci k sebe. Je to iným slovom magnet, ktorý má vplyv na objekty okolo seba. Náhlad atraktora je na obrázku 3.1. Tento magnet má nastavenú silu a od tej sily bude záležať, aký bude pomer priťahovania medzi dvoma rovnako vzdialenými `atraktormi`. Fotky, s ktorými budem pracovať, môžu mať predom nastavené parametre, ktoré určujú príslušnošť k jednotlivým atraktorom. Tieto parametre ukazujú patričnosť do kategórie, to znamená, že `atraktor` je reprezentácia virtuálnych kategórií v 3D priestore. Obrázok, ktorý patrí k danej kategórie, má nastavené desatinné číslo od nuly po jednotku,je to číslo,ktoré ukazuje, že daný obrázok akou mierou patrí do zvolenej kategórie. *Príklad*: Mám dve fotky. Na prvom sú more a západ slnka a na druhom sú len more. U prvej fotky nastavím kategórie sea a sunset, kde kategórie budú mať hodnotu $0,6$. U druhej fotky, ktorá obsahuje len more, nastavím len kategóriu sea s hodnotou $1,0$. Z tejto situácie vyplýva, že `atraktor` sea bude ťahať druhú fotku oveľa viac, ako tú prvú, ale pri `atraktore` sunset bude opačná situácia, lebo druhá fotka nepatrí do kategórie sunset, a kvôli tomu tento `atraktor` nebude mať žiadny vplyv na ňu. Tú druhú fotku bude ťahať len jeden z `atraktorov`. V prípade, že budú zapnuté všetky `atraktory`, tak prvá fotka bude umiestnená medzi tými, do ktorých patrí. Po vypnutí atraktora sunset fotky budú umiestnené okolo sea, ale tá druhá fotka bude blizšie,

lebo magnet má na ňu väčšiu silu.



Obrázek 3.1: Uzol atraktora

Obrázok môže patriť k viacerým kategóriam. U každého z nich je potrebné nastaviť parametre, teda mieru príslušnosti. Takýmto spôsobom užívateľ môže vyhľadať fotky jednoducho, podľa zvolených kritérií. Zapnutím určitých kategórií môze ovládať filtrovanie obrázkov. Po zapnutí kategórie obrázky, ktoré patria do kategórie, automaticky sa priblížia k `atraktorom`. Keď užívateľ má zapnutých viac `atraktorov` a má načítaných veľa obrázkov, môže to nesprehľadniť celú scénu. Je to spôsobené tým, že obrázky s viac kategóriami nebudú priamo u `atraktorov`, ale rozložené medzi nimi. Ako sa uskutoční rozloženie, to záleží na konkrétnom typu pohľadu s ktorým pracujeme, väčšinou je to aritmetický stred vypočítaný od `atraktorov`, k tomu je pripočítaná aj sila magnetu a príslušnosť fotky ku kategórie. Ako vidieť, veľa parametrov ovplyvňuje výslednú pozíciu. Nemusí sa vypočítať aritmetický stred, pri vzdialenosti `atraktorov` môže sa brať namiesto lineárneho rozloženia napríklad logaritmické, alebo ešte zložitejšie, ktoré si užívateľ vytvorí sám.

`Atraktory` je možné ovládať s myšou. Užívateľ môže chytiť `atraktor` a premiestniť na zvolené miesto. V prípade, že je atraktor zapnutý, fotky sa automaticky presúvajú a je vypočítaná nová pozícia pre každý obrázok, ktorý je ovplyvnený premiestneným `atraktorom`. Má to výhodu v tom, že každému je dovolené prekladať scénu tak, ako mu to najviac vyhovuje. V prípade, že scéna je už plná obrázkov a je pre neho už neprehľadná, môze to jednoducho vyriešiť: chytí `atraktor` a nastaví inú pozíciu, presunie niekam inam.

3.4 Prehliadač

Poskytnutie nového prístupu k prehliadnutiu fotiek, to je základná myšlienka. Zavedenie priťahovacích magnetov, `atraktorov`. Môj prehliadač obsahuje len najpodstatnejšie veci k tomu, aby som mohol odškúšať funkčnosť `atraktorov`, aké majú dopad na užívateľov, či je to pre nich užitočné a použiteľné. Aplikácia zahrňuje aj iné zobrazenie okrem atratorového, ale to len z dôvodu porovnávania. Bude možnosť vyskúsať si rôzne pohľady, rozmiestnenie fotiek, aby som tak našiel spôsob, ktorý mi najviac vyhovuje. Prehliadač neobsahuje žiadne prostriedky na správu dát o fotkách, to znamená, že dáta, podľa ktorých sa mi rozdeľujú fotky, musí zabezpečiť tretia strana. Tieto dáta obsahujú parametre fotiek: ktorá fotka do ktorej kategórie patrí, a navyše s akou váhou. Bez týchto údajov totiž nie som schopný určiť to, že jednotlivé `atraktory` na ktoré fotky budú mať vplyv. To, že akým iným spôsobom by sa dali získať informácie o tom, čo sa nachádza na fotkách, je v kapitole 6. Tento prehliadač umožnuje užívateľom, aby pomocou `atraktorov` vybrali len tie fotky, o ktoré naozaj majú záujem, nemusia hľadať medzi všetkými. Budú chcieť popozerať fotky o zvieratách, tak zapnú `atraktor`, ktorý sa stará o tieto fotky a vyhľadá ich. Na moje účely nie je podstatné žiadne editovanie fotiek a podobné veci, na čo som si mohol zvyknúť u klasických prehliadačoch, je to skutočne len program na prehliadnutie fotiek.

3.5 Knihovňa (modularita)

Knihovňa bola navhrhnutá tak, aby sa mohla rozdeliť na menšie časti, aby bola modulárna. Jednotlivé časti sa dali oddeliť od celku tak, aby to bolo používateľné aj bez ostatných komponentov. Boli preto vytvorené menšie balíky, ktoré obsahujú špecifickejšie triedy, napríklad tie, ktoré sa starajú o zobrazenie fotky, či tie, ktoré ovládajú scénu. K dosiahnutiu modularity je nevyhnutné, aby som dosiahol čím vyššiu úroveň abstraktnosti v mojej knihovni.

Techniky použiteľné na dodržiavanie týchto účelov:

- návrhové vzory
- abstrakné triedy
- interface – deklarácia rozhrania

3.6 Návrhové vzory

Veľmi modernú tému predstavujú návrhové vzory súvisiace s objektovo orientovným programovaním, ktoré sa zaoberajú s návrhom a vývojom informačných systémov. Návrhový vzor, ako pojem znamená obecný popis riešenia dobre známych problémov. Stále sú otázky o použiteľnosti a časovej náročnosti. Ich použiteľnosť sa môze ukázať na veľkých knihovňách, ako je aj `Qt` či `OSG`, ktoré ich používajú úspešne. Začiatok je komplikovanejší, je tu určitá réžia s vytvorením a navrhnutím, ale ten čas sa vráti neskôr, keď sa dostaneme k zložitejším veciam. Objekty, ktoré sa vytvoria s takým princípom, budú znovupoužiteľné, čo ušetrí prácu. Vymedzenie návrhových vzorov, ako popis riešenia je nepresné. Jedná sa skôr o jazyk, respektíve o spôsob komunikácie. Ich rozvoj je spojený s potrebou zachytiť možnosti riešenia netriviálnych problémov, ktoré sa opakovane objavujú pri návrhoch informačných systémov predovšetkým v oblasti dizajnu [13].

Tieto dve podmienky, tj. netriviálne a opakujúce sa situácie, predstavujú nevyhnutné podmienky pre vznik návrhových vzorov. Je zbytočné a neefektívne opisovať riešenie jednoduchých problémov, pretože v praxi sa často tieto malé problémy vyskytujú v rôznych obmenách a ich vyriešenie pomocou vedomostí a schopností softvérového špecialistu je efektívnejšie, ako pomocou návrhových vzorov. Možno povedať, že prakticky existujú dva druhy návrhových vzorov, implicitné a explicitné [14].

Implicitné sú nikde nepopísané riešenia odvíjajúce sa od znalostí a skúseností. Explicitné, ktorých je neporovnateľne menej, sú zdokumentované skúsenosti pri riešeniach konkrétnych problémov.

Druhou podmienkou je opakovateľnosť problému, aby bolo možné použiť opísané riešenie aj v budúcnosti. Nie je účelom vytvorenia návrhového vzoru publikovať riešenie špecifického problému, u ktorého je vysoká nepravdepodobnosť znovupoužiteľnosti.

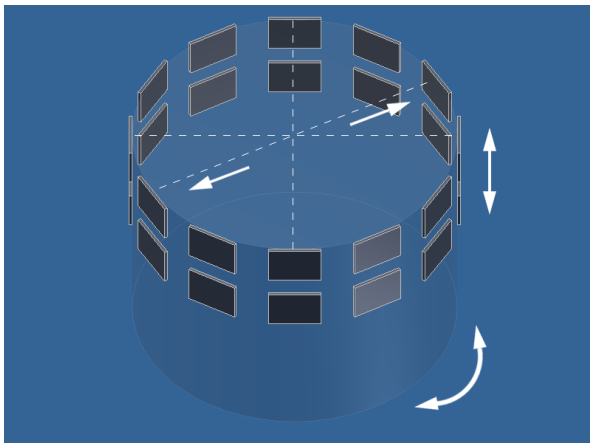
Na návrhový vzor sa nesmie pozerať, ako na osamotenú jednotku vedomostí, ale skôr, ako na výsledok kombinácie praktických a teoretických skúseností. Príkladom môžu byť návrhové vzory, ktoré vychádzajú z programovacích jazykov. Pre ich použitie musíme byť veľmi dobre zoznámení s prostredím, do ktorého bude daný vzor vsadený.

3.7 Abstraktné triedy a rozhranie

Tieto dva pojmy sú spoločné v tom, že oba definujú rozhranie tried. Keď sa implementuje do triedy interface, alebo sa zdedí od nejakej triedy, prevezme sa ich rozhranie. Abstraktné triedy sú špeciálnejšie tým, že dovoľujú okrem deklarácie operácií aj definovať ich. Trieda sa nazýva abstrakná, keď má minimálne jednu funkciu, ktorá nie je definovaná, ale to platí pre `C++` [10], v `Jave` [1] je to inak, tam trieda je abstrakná, keď je deklarovaná, ako `abstract class`. Nemusí mať ani jednu nedefinovanú funkciu. Pri implementácii interface sa musia všetky deklarované funkcie definovať, inak sa vytvorí abstraktná trieda, ktorú nie je možné inštancovať priamo. S interface a abstraktnou triedou sa dokáže zabudovať polymorfizmus a na rovnaké parametre budú triedy reagovať vlastným spôsobom. Z mojej knihovne som vybral abstraktnú triedu, ako riešenie na vytvorenie šablónov na zobrazenie fotiek. Každá trieda, ktorá bude zobrazovať moje fotky, sa musí zdediť od tej abstraktnej triedy, ktorá bude definovať funkcie, ktorých chovanie má byť vždy rovnaké. Tieto funkcie treba deklarovať, ako privátne, aby som nedovolil potomkom prepísanie týchto funkcií. Všetky zobrazenia majú rovnaké operácie, ale napríklad na načítanie obrázkov dostanem inú odozvu u každého. Ani nemusím vedieť, že inštancia, s ktorou pracujem, čo je konkrétne za objekt, stačí, že poznám jej rodiča.

3.8 Modely zobrazenia

Pri existujúcich riešeniach som rozoberal, aké spôsoby zobrazenia existujú. V knihovni navrhnem dve existujúce a dve nové riešenia. Medzi existujúce typy patria nasledujúce: špirála, valec či kolotoč, stena, kocka, guľa, aj iné, jedným slovom neopísateľné. Z existujúcich riešení vyskúšam kolotoč a stenu.

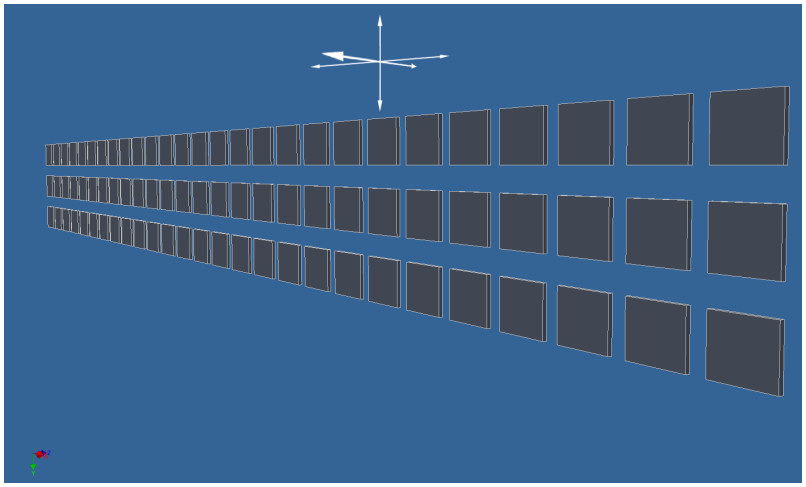


Obrázek 3.2: Schematický kolotoč

Schematické návrhy znázorňujú rozloženie fotiek v scéne a navigáciu. Akým spôsobom je môžné ovládať scénu, je nakreslené do návrhov. Nakreslené šípky ukazujú smery posúvania a rotácie.

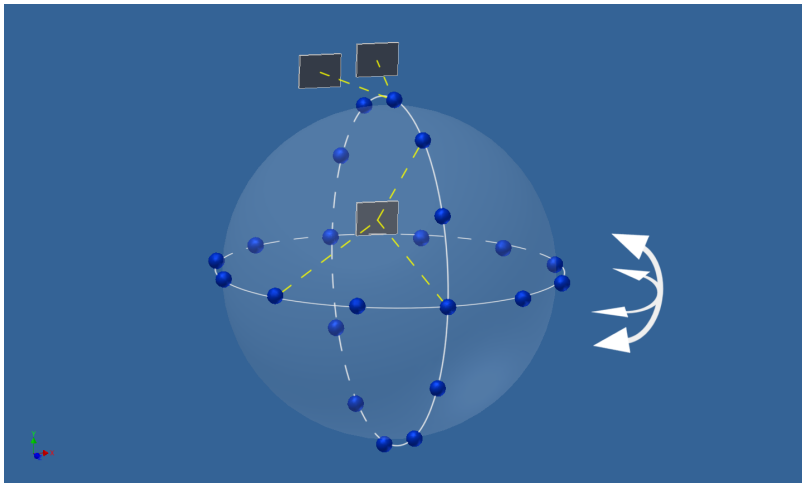
Prvý typ je kolotoč. Kolotoč patrí medzi najjednoduchšie varianty, je ľahko ovládateľný. Obrázky uložím na valec dokola, ako vidieť na obrázku 3.2. Valec sa môže otočiť len podľa osi z. Posúvanie funguje pozdĺž osi x a y.

Ďalší jednoduchý typ je 3D stena na obrázku 3.3. Mám rovinu, ktorá je rovnobežná s rovinou xz. Na túto rovinu uložím fotky, aby vytvorili mriežku. Otáčať rovinu nie je možné, funguje jedine posunutie po každej osi: hore a dole, vľavo a vpravo, dopredu a dozadu.



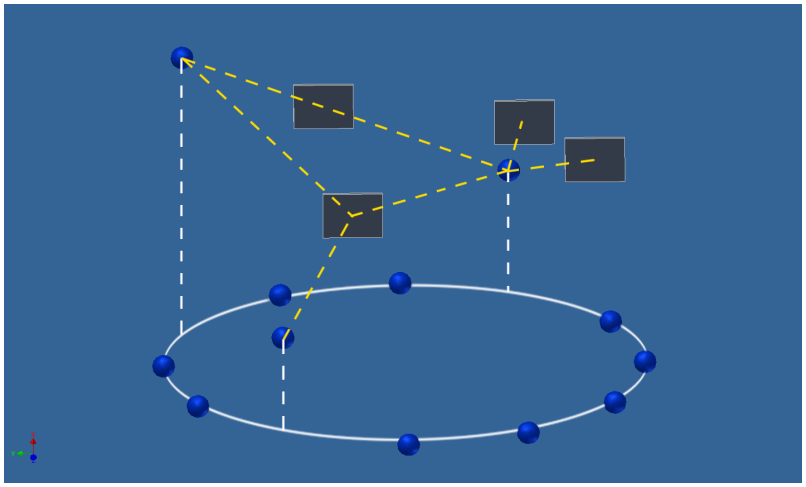
Obrázek 3.3: Schematická 3D stena

Nasleduje typ, ktorý už nebude až tak jednoduchý, ako predchádzajúce dve. Mám guľu, kde na povrhu nebudú obrázky ale `atraktory`. `Atraktory` budú rozmiestnené na dve kružnice, ktoré majú spoločný stred a sú kolmé na seba. Tieto kružnice tvoria virtuálnu guľu. Obrázky budú rozmiestnené podľa závislosti od `atraktorov`. Posúvanie tu nefunguje, je možné otočiť guľu každým smerom a približovať sa.



Obrázek 3.4: Schematická guľa s atraktormi

Posledný typ je jednoúrovňový kolotoč s `atraktormi`, obrázok 3.5. Zapnuté `atraktory` sú umiestnené vyššie, ako ostatné, tie ležia na kružnici. Rozmiestnenie fotiek je rovnako, ako u gule, závisí od polohy `atraktorov`. Ovládanie je rovnaké ako u klasického kolotoča.



Obrázek 3.5: Schematický kolotoč s atraktormi

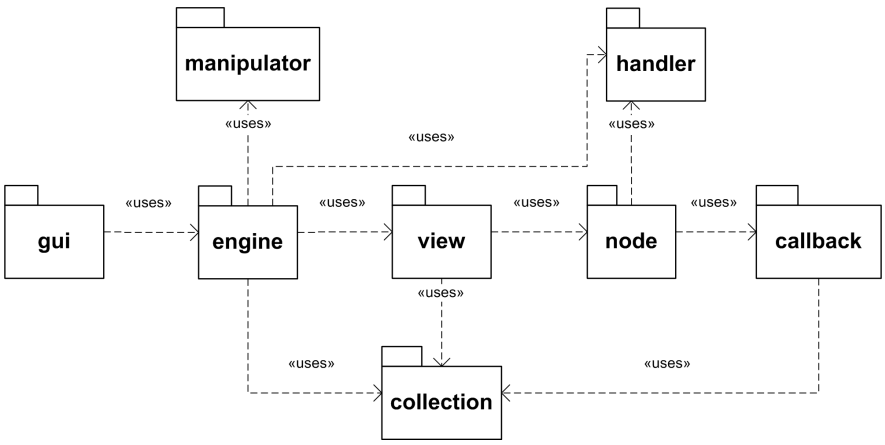
Kapitola 4

Realizácia

Táto kapitola je venovaná realizácie aplikácie. Je popísaný spôsob implementácie navrhnutých požiadavok. Obsahuje popis rozobraných častí knihovne: realizácie dynamických súčiastok a matematické výpočty s obrázkami, popis hlavných úloh jadra systému, ovládače, uzly a zobrazenie. Ako bolo písané, knihovňu som rozdelil na moduly, menšie balíky, ktoré obsahujú špeciálnejšie prvky na dané úlohy.

Balíky:

- callback (dynamika)
- engine (jadro)
- gui (grafické prvky)
- manipulator (ovládače)
- node (uzly)
- view (zobrazenie)



Obrázek 4.1: Balíky knihovne

Na obrázku 4.1 je ilustrované, ako sú prepojené jednotlivé balíky medzi sebou. To, že každý balík používa z balíka *engine* triedu `Engine`, nie je na obrázku znázornené kvôli

jednoduchosti. Jednoduchšie povedané, jadro systému potrebuje každý balík. Balík *gui* obsahuje len grafické prvky užívateľského rozhrania. Nie je to nevyhnutné, k tomu, aby jadro fungovalo, sú tam len grafikcé prvky. Bez týchto prvkov by som musel inak vyriešiť zapínanie a vypínanie kategórií. Balík engine potrebuje prvky z *view*, tie umožňujú zobraziť vybrané fotografie. V balíku *manipulátor* sa nachádzajú ovládače scény. To, že konkrétne ktorý budem používať závisí od toho, aké mám aktuálne zvolené zobrazenie. Zobrazenie dostalo miesto v balíku view. Každá trieda vo *view* používa triedy z balíka *node*, presnejšie uzly s fotkami a `atraktormi`. Na uzloch sú aplikovateľné triedy z *callback* a *handler*.

V `handler` sú dve triedy vytvorené k interakčným operáciam, detekcie kliknutia na obrázky a premiestnenie `atraktorov`. Tieto triedy napojí jadro na inštanciu `osgView::View`, čo je hlavné zobrazovacie okno v `OSG`. Posledný balík je *collection*, ktorý obsahuje všetky triedy pre prácu s kolekciami.

4.1 Dynamika

K oživeniu môjho programu pridám animované efekty, ktoré pridajú do klasického prehliadča trošku zábavy. Na tieto účely boli vytvorené triedy, ktoré sa dostali do balíka *callback*. Balík dostal meno od triedy z `OSG`. `OSG` podporuje k uzlom pripojovať callback triedy, ktoré majú virtuálnu funkciu, tá je zavolaná vždy, keď prebehol určitý dej. Balík *callback* obsahuje triedy, ktoré sú potomkami `osg::NodeCallback`. Tieto triedy sa starajú o to, aby scéna bola dynamicky meniteľná, konkrétne zodpovedajú za dynamičnosť uzlov. Potrebujem nejako zabudovať podporu animácií. Na tieto účely budú mi slúžiť spätne voliteľné triedy. Po každom vygenerovanom snímku sú zavolané tieto triedy. Výhoda je, že krokované premiestnenie sa javí plynule. Načítanie obrázkov prebieha tiež animovane, načítané obrázky sa objavia jeden za druhým a z počiatočného bodu vyletia na svoje miesto. V tejto knihovni platí rovnosť medzi slovom dynamika a callback.

Každý obraz v scéne má pripojenú svoju vlastnú callback triedu. Táto trieda je schopná riadiť pohyby obrázkov vo zvolenej scéne. Scénu nie je možné meniť mimo funkcie update, inak by som mohol spôsobiť pád celého programu. Na to treba dávať si pozor a vždy napísať svoju vlastnú triedu na aktualizáciu uzlov. Užívatelia si môžu užívať animácie s podporou týchto tried. Na callback triedy sa môže myslieť ako na vlastne definované funkcie, ktoré sú automaticky vykonávané v závislosti na typu prechodu (update, cull, draw). Callback-y môžu byť spojené s jednotlivými uzlami, alebo môžu byť spojené s vybranými typmi (alebo podtypy) uzlov. Počas prehodu scény, ak sa u uzlov vyskytol callback, bude zavolaný a vykonaný. V balíku sú triedy napísané, ako updade callback triedy, budú zavolané vždy pri prechode aktualizácie grafu.

Samozrejme môj zdroják nemusím písať do týchto tried, nie som nútený vytvoriť callback triedy. Môžem umiestniť aj v hlavnej smyčke medzi update a cull, kde sú generované snímky, ale takým spôsobom sa lepšie udržuje prehľadnosť zdrojákov a navyše mám zapuzdrené veci, ktoré môžem skryť pred inými užívateľmi.

Zapuzdrenie má dobré i zlé stránky, podobne ako skoro všetko, ale má oveľa viac pozitívnych, ako negatívnych. Tým, že skryjem kód, sa zabezpečím proti tomu, aby mi niekto úmyselne či neúmyselne prepísal algoritmus. Môžem si byť istý, že vytvorený kód bude vždy robiť to, čo má. Vidieť flexibilitu týchto tried, môžem ich vymeniť, alebo používať viac naraz, lebo všetky sú zdedené od spoločnej rodičovskej triedy.

Balík obsahuje triedy, ktoré sú pripojiteľné na všetky pohľady, to je ich výhoda, ale nemusia sa chovať rozumne u každého z nich. To sa nastane v prípade, keď používam nejaké zobrazenie, kde definované pohyby v callback triede sa budú chovať tak, že pri kliknutí mi

kamera vyletí hore a nie je to najlepšie v prípade, že fotky mám iným smerom. Je to len príklad, ale bez problémov sa to dá vyriešiť.

Callback triedy sa starajú aj o to, aby pri kliknutí na obrázky sa načítal veľký, poprípade malý obrázok, to záleží od konkrétnej situácie. Ďalej majú za úlohu vypočítať bod, do ktorého musíme premiestniť obrázok, keď na to užívateľ klikol, aby sa to javilo, že sa to zväčšilo, a po kliknutí sa automaticky vrátilo na pôvodné miesto. Musí sa ešte vypočítať aj uhol otáčania, lebo obrázok nemusí byť otočený smerom k nám, to znamená, že normála obrázky a normála kamery nie sú súbežné. Situáciu ilustruje nasledujúci obrázok.

Ako vidieť, obrázok môže byť naklonený podľa troch ôs, x, y a z, a musím vypočítať rozdiel naklonenia s kamerou. Obrázok nemôžem umiestniť priamo do pozície kamery, tam tiež je potreba vypočítať si bod, ktorý leží medzi kamerou a obrázkom, ale v takej diaľke, aby bolo celé viditeľné. Tu potrebujem aj pomer strán okna. Vypočítať potom konecnú pozíciu nie je už komplikované, je to čistá analytická geometria. Súradnice týchto bodov poznám, spojím ich a mám hotovú analytickú rovnicu priamky. Teraz potrebujem uhol pohľadu kamery a pomer strán zobrazovacieho okna. Pomocou týchto údajov už viem vypočítať pozíciu, ktorá bude ležať na priamke medzi kamerou a východiskovým bodom. Moje callback triedy sú schopné tieto pohyby uskutočniť animovane. Pri detekcii kliknutia vypočítajú požadovaný bod a postupne posúvajú obrázok na jeho nové miesto, čo je jednoduché kvôli tomu, lebo mám analyticky vypočítanú rovnicu, kde potrebujem nastaviť len jeden parameter, ktorý leží medzi nulou a jednotkou. Tento interval rozdelím podľa toho, akú rýchlosť potrebujem. Budem mať napríklad 100 snímkov za sekundu, interval rozdelím na tisíc častí, bude to trvať 10 sekúnd, čo je veľa, musím si dávať pozor, aby to netrvalo príliš dlho. Ďalšie efekty je možné pridať jednoducho do úseku, kde je parameter krokovaný. Ja tu mám taký efekt pridaný, keď sa obrázok približuje, otočí sa podľa nastavených ôs, kým parameter nedokrokuje obrázok do konečnej pozície.

Na skrývanie a animované zapnutie boli vytvorené dve triedy, ktoré pracujú s `atraktormi`. Prvá trieda automaticky skryje uzol `atraktoru`, keď daná kategória nie je aktívna. Túto callback triedu používam len u jedného pohľadu, u druhého by sa dalo tiež, ale sú situácie, keď to nedáva zmysel. Druhá trieda pracuje len u druhého pohľadu, má funkciu zdvíhať `atraktory` hore, keď boli zapnuté, a pri vypnutí automaticky ich vráti na pôvodné miesto.

4.2 Jadro systému

Na riadenie systému je dôležité vytvoriť jednotku, ktorá bude spracovávať najpodstatnejšie veci, generovať udalosti o zmene stavu a podobné veci. Táto jednotka je jadro systému, ovláda jednotlivé typy zobrazenia, generuje signály pri udalostiach, načíta mená obrázkov z adresára, predáva CSV súbory na spracovanie, z čoho vytvorí hotovú kolekciu, s ktorou sú schopné pracovať jednotlivé zobrazenia.

Pri tvorbe som používal návhrhové vzory. Táto trieda bola navrhnutá ako jedináčik (*Singleton* [14]), je možné vytvoriť v priebehu programu len jednu inštanciu, a tá prvá musí byť inicializovaná s inštanciou triedy `osgViewer::View`, ktorá nám zobrazí celú scénu. Triedna má statickú funkciu, ktorá slúži na získanie inštancií, preto súčasne môže existovať len jedna. Potrebujeme pracovať s Engine, alebo získať informácie, napríklad o stave načítania obrázkov, stačí zavolať továrenskú metódu, ktorá mi to vráti.

Jadro systému sa nachádza v balíku *engine*. Je to menší balík, obsahuje len dve triedy, ktoré sú duchom programu a bez nich by boli ostatné triedy nepoužívateľné. Po vybraní adresára alebo CSV súboru sa automaticky začne spracovanie dát. Jadro je implementované

ako vlákno, aby pri načítaní nezablokovalo ostatné prostriedky. Ak bol zvolený adresár, tak načíta obrázky do zoznamu a predá ich na spracovanie k aktuálne vybranému zobrazeniu.

Má zabudovaný pseudo-random generátor, ktorý generuje pseudo-náhodné čísla. Tieto čísla sú využité hlavne u `atraktorov` v prípade, že viac obrázkov padne na to isté miesto, a aby sa vyhlo prekrývaniu, sú pozície prepočítané pomocou týchto náhodných čísiel. Výhoda, že `Engine` je jedináčikom, je v tom, že každý objekt môže získať informácie o stave načítania obrázkov, aktuálne zobrazenie, dáta od kamery, alebo sa napojiť na signály a pozorovať jednotlivé udalosti, tak ako to robia aj prvky grafického užívateľského rozhrania. Po zmene kategórií je generovaný signál o tom, že sa zmenili kategórie, a objekty, ktoré sú napojené na tento signál, ich môžu aktualizovať, pokiaľ to potrebujú.

Jadro nám dovoľuje vybrať si z viacerých typov zobrazenia a nastaví aj potrebné ovládače k zobrazeniam, ktoré umožnujú jednoduchšie ich ovládať. Keď zvolím nové zobrazenie, aktuálne vybrané bude deaktivované, vymením ovládač, aktivujem nové zobrazenie a potom začne znova načítať obrázky a vytvoriť scénu podľa vlastného nastavenia. Funkčnosť a použitie týchto ovládačov je podpobnejšie popísané v kapitole 4.4.

Jadro má na seba napojené manažéry, ktoré obsluhujú výber a ťahanie špeciálnych uzlov v scéne.

Balík obsahuje ešte jednu triedu, ktorá vytvorí grafický kontext pre obrázky v scéne a nastaví potrebné parametre. Triedy tohto balíka patria medzi najdôležitejšie časti celého systému.

4.3 Interaktivita

Správa interakcií je dôležitou súčasťou systému. V balíku *handler* sú dve triedy, ktoré spravujú interakcie s okolitým svetom. Užívatelia ovládajú program s myšou, tak aby mohli obrázky zväčšiť, alebo premiestniť magnety, aby vytvorili dve triedy: `PickHandler` a `DragHandler`. Ako aj ich mená ukazujú, slúžia na detekciu kliknutia a posunutie uzlov. Tieto manažéry sú napojené na jadro systému.

Predstavme si situáciu, že máme načítané fotky a prehliadame ich. Vyberieme jednu, ktorú chceme pozrieť vo väčšom.

Máme dve možnosti:

- priblížime sa k celej scéne
- priblížime obrázok k nám

Prvá možnosť sa môže realizovať aj bez podpory týchto manažérov, ovládače dovoľujú priblíženie, ale na také účely je to obtiažne. Riešenie prinášajú správcovia interakcií, detekujú kliknutie v scéne. Ak došlo k interakcii s obrázkom alebo magnetom, bude odoslaná správa k príslušnému uzlu. Uzol už podľa seba zariadi spracovanie.

Pri kliknutí často sa stáva, že obrázky sa prekrývajú bez toho, aby o tom užívateľ vedel. Z toho dôvodu treba nájsť uzol, ktorý sa nachádza nad ostatnými. Detekcia prebieha tak, že pomocou súradníc zobrazovacieho okna sú vypočítané priesečníky a vyhľadané všetky uzly, nad ktorými bol aktuálne kurzor [8]. Správca je napísaný tak, aby spracoval uzol, ktorý je najblizšie k obrazovke, teda smerom k užívateľovi. Ostatné uzly sú tiež detekované, keď sa prekrývajú, ale sú ignorované, lebo užívateľa zaujíma len ten najbližší. Nebolo by dobre, keby sa aktivovali aj také obrázky, ktoré sú za vybraným. Teoreticky to ide, ale logickejšie je ostatých ignorovať, aby sa takto simulovala realita.

Užívateľ vidí presne obrázok nad ktorým sa nachádza, na to je zabudovaná podpora. Tieto obrázky sú zvýraznené tak, že sú vo zväčšenej forme oproti ostatnými. Obrázky pri nabehnutí kurzora sa zväčšia a po opustení sa zmenšia na pôvodnú veľkosť. Táto drobnosť bolo pridaná, aby užívatelia mali nejakým spôsobom vyznačené, nad ktorým obrázkom sa aktuálne nachádzajú.

Ak bolo detekované, že užívateľ klikol na uzol s obrázkom, `PickHander` zistí, či ide o dvojklik s ľavým tlačítkom. V prípade, že áno, tak podľa primitívy nájde konkrétny uzol ku ktorému patrí. Ak je to uzol obrázku, tak je aktivované načítanie originálneho obrázku namiesto miniatúry. Vypočíta sa nová pozícia pre presunutie, lebo ide o priblíženie. Opačne to funguje podobne, vymení sa naspäť na miniatúru a posunie sa na pôvodné miesto.

`DragHandler` funguje dosť podobným spôsobom, ako `PickHandler`, tiež vypočíta priesečníky a vyhľadá uzly, ale teraz je potrebné vyhľadať uzly typu `osgManipulator::Dragger`. Tieto uzly sú zabalené do iných špeciálnych uzlov tak, aby ich bolo možné premiestniť. Vždy je potrebné prepnúť sa do módu, ktorý umožnuje premiestniť tieto uzly. Prepnúť sa do toho režimu je možné s klávesovými skratkami. V tomto módu užívateľ môže chytiť zvolený `atraktor` a premiestniť na vhodné miesto. Je potrebné nastaviť premiestniteľným uzlom naslúchanie interaktívnych akcií. Obrázky patriace k premiestneným magnetom ho automaticky budú nasledovať. Tento manažer je aplikovaný na uzly, ktoré sú typu AttractorNode. Obrázky nie je možné nijakým spôsobom premiestniť samostatne.

4.4 Ovládače

K rôznym zobrazeniam sú potrebné rôzne ovládače. To znamená, že nie všetky zobrazenia sa ovládajú rovnako. Máme tu všelijaké typy: kolotoč, stena, guľa. K týmto zobrazeniam máme vytvorené špeciálne ovládače, ktoré sú špecifické na konkrétne typy. Sada ovládačov bola umiestnená v balíku manipulátor. Ako aj meno ukazuje, tento balík obsahuje všetky triedy, ktoré sú zodpovedné za ovládanie scény. Tieto triedy sú aplikovateľné na zobrazenie z balíka view. Je tu navyše jedna trieda, ktorá slúži na manipuláciu `atraktorovými` uzlami.

K jednotlivým pohľadom boli vyrobené špeciálne ovládače, aby scénu s obrázkami mohol užívateľ manipulovať čím jednoduchšie, kvôli tomu sú zablokované niektoré smery otočenia [9]. Nie je vždy potrebné dovoliť, aby užívateľ mohol otáčať scénu podľa všetkých ôs, veľká voľnosť je niekedy väčšia prekážka. Ovládač spravený pre zobrazenie s `atraktormi` v guli je jediný, ktorý umožnuje otáčať všetkými smermi.

Ovládač trojdimenzionálnej steny má jednu špeciálnu funkciu: funguje tak, že keď máme priblíženú stenu a pohneme sa niektorým smerom, manipulátor nás automaticky obráti a posunie nás po stene. Vyzerá to, ako by sme leteli blízko steny. Keď sa zastavíme, otočí nás do pôvodnej polohy.

Kamera v `OSG` má prehodené súradnice, preto je lepšie sa opýtať na práve používaný manipulátor, keď sú potrebné dáta kamery, aktuálna pozícia, normála, atď. Callback triedy tieto údaje používajú na výpočet posunutia obrázkov pred kameru . Tento ovládač dovoľuje najmenej voľnosti pri pohybe. Umožňuje len pohyb smerom hore, dole, doľava, doprava, a priblížiť sa, všetky rotácie sú zakázané.

Posúvač `atraktorov` je jediný z ovládačov, ktorý má niečo viditeľného v scéne. U mňa je to guľa, do ktorej je zabalený `atraktorový` uzol. Pomocou tejto gule je možné premiestniť ich ľahko a jednoducho. `OSG` obsahuje aj svoje vlastné triedy na podobné účely, ale ani jedna z nich nebola tá najvhodnejšia. Umožnovali nielen posúvanie, ale aj zväčšiť a zmenšiť posúvače, čo nepotrebujem pri práci,zkomplikovalo by prácu užívateľom.

4.5 Špeciálne uzly

Node je balík obsahujúci dva typy špeciálnych uzlov, ktoré používa moja knihovňa. Prvý je uzol, ktorý spravuje obrázky, druhý je už vyššie spomenutý posúvací uzol. Tieto uzly boli vytvorené z dôvodu vizualizácie `atraktorov` a obrázkov v scéne.

Medzi základné operácie, ktoré musím realizovať patria: normalizácia rozmerov fotiek, zväčšenie, otáčanie podľa lokálnych ôs, výmena kvality.

Základ mojej scény tvorí uzol s obrázkom (`ImageNode`). Táto trieda zapuzdruje všetky podstatné operácie, ktoré súvisia s obrázkami.

Načítanie:
Uzol najprv zistí, či existuje súbor s obrázkom. Ak áno, potom skúsi vytvoriť miniatúru v prípade, že ešte neexistuje. Je to potrebné preto, aby náš program bol rýchlejší a nezabral príliš veľa miesta v pamäti. `OSG` má podporu na uvoľnenie pamäti po texturách aplikácie, ale v praxi to nefungovalo. Keď už sú vytvorené miniatúry, čas potrebný na načítanie je kratší. Miniatúry sú približne desaťkrát menšie, čo sa týka velkosťi súborov, preto proces načítania prebehne tiež desaťkrát rýchlejšie. Je to výhodnejšie, užívateľ nemusí toľko čakať. Pri načítaní bude program blokovať prostriedky, preto je lepšie, keď budem implementovať načítaciu operáciu podobne, ako je implentované v jadre, ktorý sa spustí ako nový proces; takto sa zabráni blokovaniu.

Normalizácia:
Rozmery obrázkov sú vždy normalizované, aby sa mohlo pracovať jednodušie, keď sa umiestnia do scény. To znamená, že existuje maximálna veľkosť, čo obrázky nemôžu prekročiť. Poznám maximálne hranice a musím kontrolovať,či ich nepresahujú, ak áno, musím vykonať zmenu rozmerov. Pomocou tejto úpravy nebudem mať veľmi malé a veľmi veľké obrázky, ale budú medzi určitými hranicami.

Môj uzol je poskladaný z rôznych menších častí. Textura je nanesená na dva trojuholníky, ktoré sú poskladané z primitívov, tieto tvoria geometriu obrázkov. Táto geomertia je pridaná do uzlu osg::Geoge, ktorý to vykreslí. Tento musím zabaliť ďalej do uzlu, ktorý má meno `osg::PositionAttituteTransform`, čo je transformácia závislá od polohy [7]. Bude mi to slúžiť na to, aby som mohol obrázok otočiť podľa vlastých ôs a nie podľa globálnych. Posledný baliaci uzol, je `osg::MatrixTransform`, z čoho sa zdedí aj trieda. Má zabudované operácie potrebné k otáčaniu, posunutiu, atď.

Uzol pre atraktor je tiež poskladaný z menších častí. Tento je už jednoduchší. Vytvorí mi `atraktor` podľa zadaného mena a posúvač podľa zvoleného typu. Umožňuje používať aj preddefinované posúvače, ktoré poskytuje `OSG`, ale môžem zvoliť aj posúvač, ktorý obsahuje moja knihovňa. Do uzlu je pridané meno `atraktoru` ako text, aby bolo jednoduchšie rozpoznať ich, navyše každý má inú náhodnú farbu kvôli rozpoznateľnosti. Tieto uzly sú vytvorené tak, aby ich text sa otočil vždy smerom ku mne pri manipulácii so scénou. Obalom je guľa z mriežky. Tento tvar pri otáčaní bude vyzerať vždy rovnako a pri premiestnení sa dá chytiť najjednoduchšie. Náhľad je na obrázku 3.1.

4.6 Kolekcie

K práci s obrázkami potrebujem spraviť štruktúru na informácie. Tieto informácie budem čerpať z CSV súborov. Na také účely bol vytvorený balík *collection*. Sú v ňom triedy, ako `Category`, ktorý obsahuje všetky informácie o danej kategórii, ďalej je tu ImageParams, čo je implementovaná ako mapa a slúži na uloženie parametrov k obrázkom, umožňuje mi

jednoducho nastavovať a získať hodnoty. Obsahuje aj triedu na interpretáciu CSV súborov. Táto trieda predpokladá, že na vstupu je validný súbor. Prvý stĺpec musí byť cesta k obrázku, absolútna alebo relatívna, ostatné stĺpce obsahujú hodnoty parametrov.

4.7 GUI

Aby som vedel zapnúť a vypnúť kategórie, zmeniť zobrazenie, nastaviť silu `atraktorov`, potrebujem prvky grafického užívateľského rozhrania. Ako už bolo písané, prvky užívateľského rozhrania mi poskytne knihovňa `Qt`, ktorá je na to špecializovaná. Základné prvky už existujú, ja budem musieť tieto len rozšíriť.

Balík obsahuje menšie triedy (panel, gombíky) a jednu väčšiu (okno), ktoré slúžia na interakciu s užívateľmi, inými slovami, triedy grafického užívateľského rozhrania. Mám tu dva typy panelov, z ktorých prvý slúži na prepínanie medzi zobrazeniami a druhý na zapnutie a vypnutie `atraktorov`. Tieto panely obsahujú tlačítka, ktoré sú napojené na jadro systému, aby dostali informácie o zmenách. Pri zapnutí `atraktorového` tlačítka je oznámené jadru, že kategórie boli zmenené, preto musí vykonať určité obslužné operácie. Je tu ešte možnosť nastaviť aj silu `atraktorov`, čím je číslo väčšie, tým viac priťahuje k sebe obrázky. Tlačítka druhého panela oznámia, že užívateľ zvolil iné zobrazenie, čo jadro musí tiež obslúziť. Pri týchto užívateľských akciách sú generované signály na základe knihovne `Qt` a sú definované obslúžne sloty.

Sem patrí aj hlavné okno aplikácie, čo obsahuje všetky zmienené triedy a má nastavené aj horné menu na otvorenie adresárov, či CSV súborov.

Most medzi knihovňami `Qt` a `OSG` tvorí trieda `QOSGWidget`. Táto trieda dostane všetky udalosti od užívateľov, preto ich musí poriadne obslúžiť a predať potrebné informácie k `OSG`, aby vedel spracovať, inými slovmi emulovať ich. Detekuje nielen kliknutie, tlačenie tlačítka, uvoľnenie, ale aj zmenšenie a zväčšenie okna a podľa toho nastaví perspektívu zobrazenia. Okno `OSG` je zabalené do okna `Qt`, lebo `Qt` knihovňa má oveľa lepšie užívateľské rozhranie, dopredu napísané okná, zoznamy, panely, dialógové okná.

4.8 Zobrazenie

Viem, že budem vytvárať viac zobrazení. Je to dôvod, aby som sa pokúšal spraviť základnú šablónu, z ktorej budú ostatné vychádzať. V balíku dostali miesto triedy, ktoré mi zobrazia všetko, čo vidí užívateľ. Každá sa zdedí od spoločného rodiča, ktorý má dopredu definované operácie. Tieto operácie potrebuje každá trieda. Implementácia by bola rovnaká, preto je používaný spoločný rodič a nie len interface, ktorý by definoval rozhranie tried. Takto som zabezpečil, aby moje triedy sa správali rovnako pri rovnakých operáciach. Umiestnenia obrázkov musia zobrazenie implementovať vlastným spôsobom. Na to majú čiste virtuálne funkcie, ktoré musím prepísať pri definícii triedy. V `C++` čiste virtuálna funkcia znamená, že má len deklaráciu, ale to nie je nijakým spôsobom definované. Také triedy sa nazývajú abstraktné triedy, lebo z nich nie je možné priamo vytvoriť inštancie. Je nutné z nich zdediť, ak chcem používať ich dopredu definované operácie. Takto som dosiahol šablonovanie zobrazení.

Abstraktná trieda v balíku je `View`, z nej sú vytvorené ostatné špeciálnejšie typy. Základ funguje tak, že aj táto trieda sa tiež zdedí, ale táto od `osg::NodeCallback`, čo umožňuje, aby typy zobrazení sa pripojili na nejaký uzol, čo bude tvoriť základ scény, do ktorého budem vkladať moje uzly s načítanými obrázkami. Pridávanie obrázkov je spravené tak, že

trieda má zásobník, do ktorého uložím aktuálne načítaný obrázok. Keď `OSG` prejde moje uzly a zavolá túto callback triedu, vyberie zo zásobníka obrázok, keď v ňom čaká nejaký a pridá do scény. Aby sa to naozaj uskutočnilo, musím mať povolené moje zobrazenie.

Môže sa to predstaviť tak, že je jeden spoločný koreň, či základný kameň. Tento základný kameň má na seba pripojené všetky zobrazenia. Tieto zobrazenia sú na začiatku vypnuté, a zapnuté môže byť súčasne len jedno.

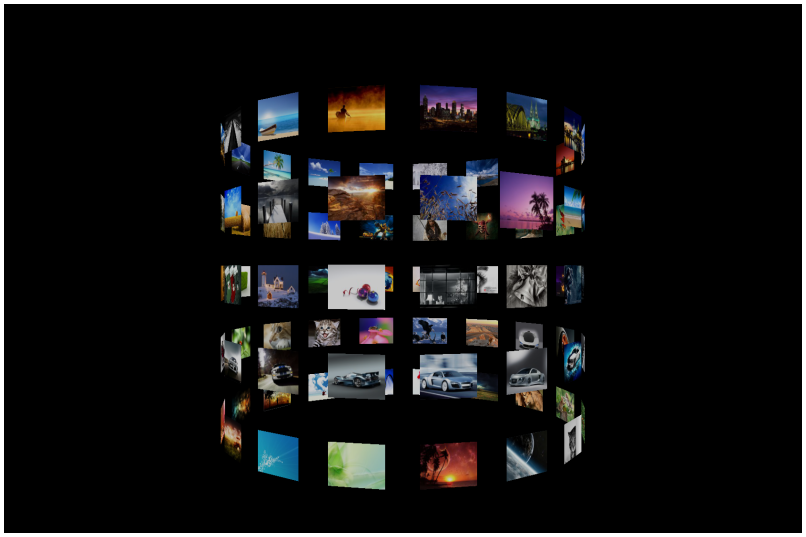
Tento callback je možné povoliť a zakázať s jednoduchým spôsobom: treba zavolať obslužné funkcie. Trieda signalizuje stav hotovo, ak načítanie bolo ukončené a všetky obrázky boli pridané do scény. Čo je ešte dôležité, obsahuje funkciu na nastavenie vzdialenosti kamery, aby po pridaní obrázkov bolo vidieť celú scénu – treba nastaviť "bounding sphere". Trieda má dve dôležité virtuálne funkcie s rovnakým menom load, jednému stačí predať zoznam s menami obrázkov, druhý potrebuje kolekciu. V každom potomku by mal implementovať túto funkciu takým štýlom, že nastaví príznak načítania, prejde zoznam fotiek či kolekciu, vytvorí nový obrázok, pridá k tomu callback triedu, ktorá definuje jeho chovanie a nakoniec pridá to do zásobníka. Keď je napríklad 100 obrázkov, ich pridanie do scény po načítaní – už sú v zásobníku –, bude trvať 100 snímkov, lebo funkcia, ktorá ich pridáva do scény je zavolaná raz behom generovania jedného snímku.

Kolotoč

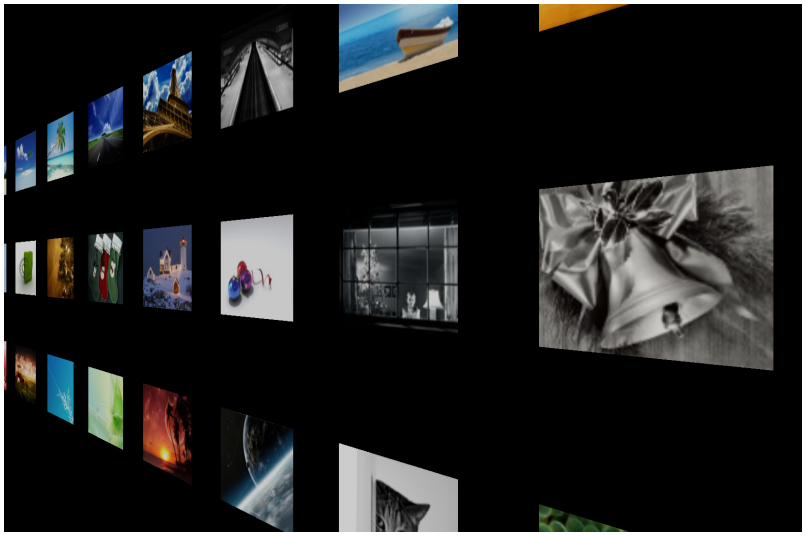
`CarouselView` patrí medzi najjeduchšie typy zobrazení obrázkov v 3D. Používajú to na veľa miestach, aj programoch s podporou 3D akcelerácie grafickej karty, ale aj bez toho. Napríklad všelijaké flash aplikácie, ako galérie fotiek, alebo aj zásuvný modul do prehliadačov webových stránok (*Internet Explorer* či *Firefox*). Tento typ rozloží moje fotky štýlom kolotoča. Kolotoč je poskladaný z viacerých krúžkov, ktoré obsahujú predom definovaný počet obrázkov. V prípade, že mám menej obrázkov, koľko by stačilo na jeden krúžok, obrázky sú rozdelené tak, aby boli rovnomerne rozložené. Dvojitým kliknutím sa obrázok priblíži ku mne a dvojitým kliknutím sa zase vráti na pôvodné miesto. Užívateľ môže popozerať fotky zvonka alebo priblížením zvnútra kolotoča, podľa toho, ktorý spôsob mu viac vyhovuje. Táto scéna má svoj vlastný ovládač, ktorý dostal miesto v balíku manipulátor pod názvom `CarouselManipulator`. Umožnuje otočenie podľa osi z a posúvanie paralelne s osou z alebo y, jedným slovom, približovať a posúvať hore či dole.

3D stena

`WallView` je tiež na veľa miestach používaný typ. Podobne funguje aj prehliadač dostupný pre telefóny *iPhone* od firmy *Apple*, rozdieľ je, že sa tam ovláda aplikácia s dotykovým displejom. Podobný je aj *Cooliris* zásuvný modul do *Firefox*. Fotky tu budú rozložené na 3D stene. Táto stena reprezentuje niečo medzi 3D a 2D, hoci sa pracuje v trojrozmernom prostredí, fotky sú nahodené na rovinu, ktorá je dvojrozmerná. To, že je to naozaj 3D, vidieť len vtedy, keď sa pohybujeme pozdĺž steny. Stena sa automaticky nakloní a kamera sa posúva, čo vnímame, ako by sme leteli pred ňou. Za tento efekt je zodpovedný ale ovládač, ktorý treba aplikovať spolu so scénou. Toto zobrazenie rozloží fotky vždy rovnako, nezáleží na tom, či sa zvolí adresár alebo kolekciu. Aj tu funguje dvojklik na obrázky podobne ako u kolotoča. Rozdiel je v tom, že tu sa približuje k obrázkom kamera. Ovládač určený k tomu typu neumožnuje žiadne otáčky okolo ôs, jediné čo funguje je posun hore a dole, pohyby doprava a doľava a zväčšovanie.



Obrázek 4.2: Zobrazenie typu kolotoč



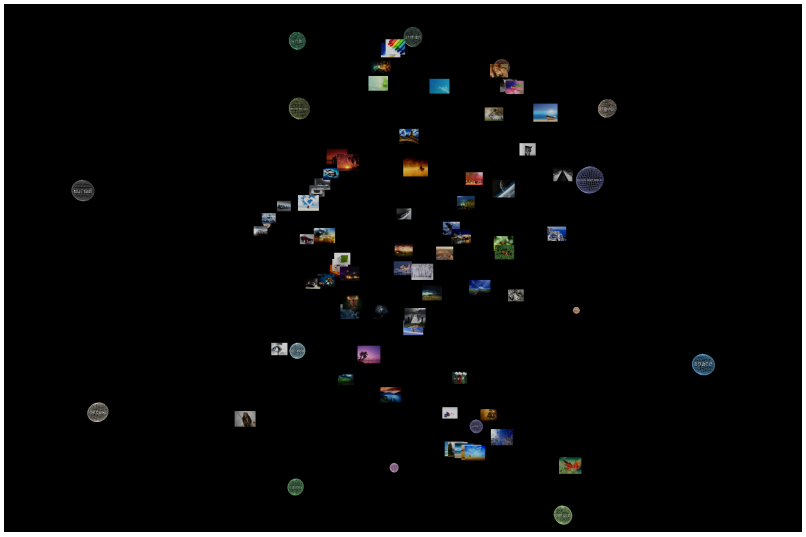
Obrázek 4.3: Zobrazenie typu 3D stena

Guľa s atraktormi

Tento princíp zobrazenia obrázkov je oveľa komplikovanejší, ako predchádzajúce dva. To sa týka nielen implementácie, ale aj ovládania princípu užívateľmi. To, že na prvý pohľad je náročnejší ešte neznamená, že nemá výhody. Tento typ funguje len s kolekciami, lebo potrebuje parametre obrázkov, aby vedel nastaviť umiestnenie medzi `atraktormi`. Zoznam, ktorý je vyrobený z načítaných mien súborov neobsahuje žiadne informácie, ktoré tieto `atraktory` potrebujú na to, aby sa mohli rozumne rozložiť, naviac pracovať s nimi. Atraktory tvoria guľu, presnejšie hraničné body gule. Obrázky sa nachádzajú implicitne vždy medzi tými bodmi. Scénu môžem podľa chuti prerobiť, stačí len zapnúť mód, v ktorom je dovolené premiestniť `atraktory`. Ak začnem tahať jednotlivé `atraktory`, obrázky sa automaticky

začnú posúvať tým smerom, akým bol `atraktor` posunutý.

Načítané obrázky sú umiestnené do centrálneho bodu. Odtiaľ budú premiestnené, keď bude vypočítané ich správne miesto. Výpočet je nechaný na callback triedy, tie sú zodpovedné za tieto účely. Proces načítania sa bude líšiť od predchádzajúcich prípadov len tým, že tu musím vyrobiť aj potrebné `atraktory`, ktoré zistíme z aktuálnej kolekcie. Kategórie sú implicitne zapnuté, takže všetky `atraktory` sú aktívne na začiatku. Vypnúť a zapnúť kategórie je možné na bočnom paneli. Po vypnutí sa obrázky automaticky preskupia, ak obrázok má vypnuté všetky kategórie, sa skryje. Zapnem znova `atraktory`, obrázky sa znovu objavia. U obrázkov, ktoré sú skryté, nie sú zavolané callback triedy, preto musím ich nájsť a znova ich aktivovať zvonka. To podobne funguje u `atraktorov`, len tam nemusím nič vyhľadať, presne viem, ktoré treba znova aktivovať. Je to podobné tomu, akoby som zatvoril dvere, ktoré je možné otvoriť len zvonka.

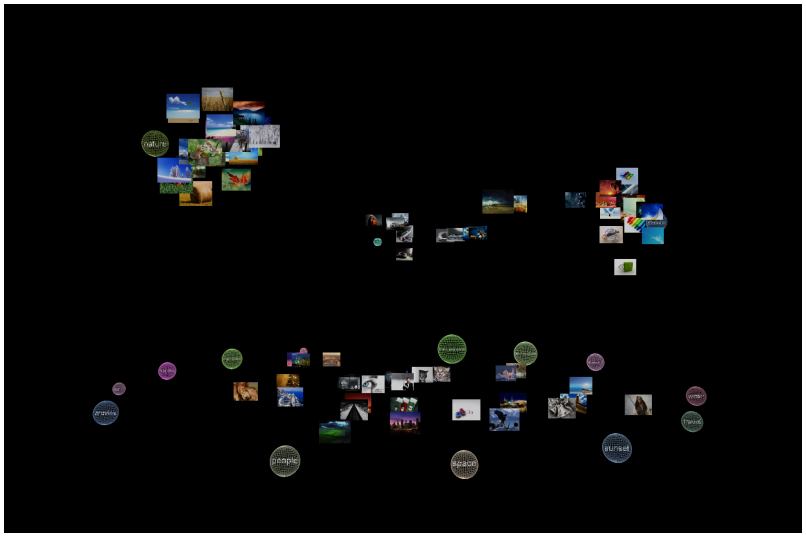


Obrázek 4.4: Zobrazenie typu guľa s atraktormi

Kolotoč s atraktormi

Konečne sme sa dostal k tomu poslednému pohľadu. Tento pohlad je prehľadnejší, ako ten druhý variant. Podobá sa to na kolotoč, lebo `atraktory` sú rozmiestnené dokola. Na začiatku sú všetky implicitne vypnuté. Obrázky sú náhodne rozsypané medzi nimi. Zapnutie `atraktorov` funguje rovnako ako predtým, ale teraz, keď mám vypnutý `atraktor` nie je skrytý. Vypnuté `atraktory` sú umiestené na základnej kružnici. Po zapnutí sa `atraktor` zdvihne na náhodne vysoké miesto a obrázky, ktoré k tomu patria sa tiež automaticky zdvihnú na svoje nové vypočítané miesta. Premiestniť `atraktor` je aj tu možné, tiež musím to povoliť pomocou klávesnice. Výpočet a posunutie obrázkov je tu oveľa zložitejší a náročnejší na výpočetný výkon. Musím prepočítať každú pozíciu trikrát.Najprv vypočítam centrálu `atraktorov`, to ešte nie je tažké. Prejdem všetky `atraktory` ku ktorým aktuálny obrázok patrí a sčítavam len x-ové súradnice s x-ovými a takisto aj ostatné súradnice. Nakoniec vydelím ich s počtom `atraktorov` pre daný obrázok a mám stredný bod. Obrázok môže mať rôzne váhy príslušnosti ku kategóriam, preto vypočítam novú pozíciu s novými parametrami. Teraz vychádzam už z vypočítaného bodu. Kvôli tomu, že váhy `atraktorov`

sú nastaviteľné, potrebujemeprepočítať pozíciu s váhami. Mám bod, ktorý som potreboval. O výpočte podrobnejšie v kapitole 4.9.



Obrázek 4.5: Zobrazenie typu atraktorový kolotoč

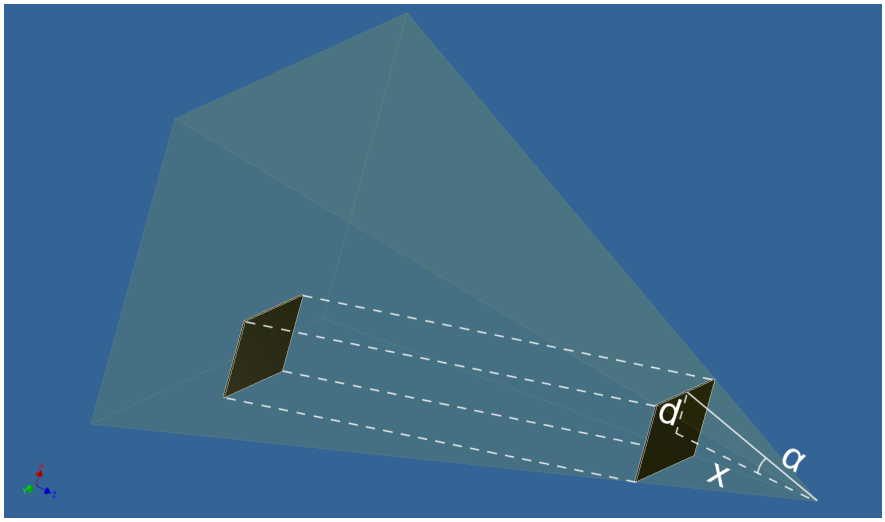
4.9 Dynamika - priblíženie fotiek

Mám body $A[x_a, y_a, z_a]$ a $B[x_b, y_b, z_b]$, vektor $\vec{v} = B - A$ a parameter $t \in < 0..1 >$. Chcem vypočítať priamku, ktorá ide od bodu A do bodu B, to je možné pomocou rovnicovej sústavy 4.3. Bod A reprezentuje pozíciu fotky a bod B pozíciu, kam umiestnim fotku, teda novú pozíciu. Fotku nemôžem umiestniť priamo do bodu, kde sa nachádza kamera. Musím vypočítať vzdialenosť, kde bude ešte celá fotka viditeľná.

$$\begin{aligned} x &= x_A + t * x_{\vec{v}} \\ y &= y_A + t * y_{\vec{v}} \\ z &= z_A + t * z_{\vec{v}} \end{aligned} \tag{4.1}$$

K animácii stačí len krokovať parameter t v definovanom intervale, čísla x, y a z mi dávajú aktuálne súradnice vypočítaného bodu. Ak $t = 1$ rovnica mi vráti presne bod B, triviálne pre $t = 0$ bod A. K výpočtu potrebujem uhol viditeľnosti ($fovy$) a stranový pomer ($aspectRatio$). Poznám výšku a šírku fotky, podľa toho zostavím dve rovnice, s ktorými som schopný vypočítať vzdialenosť, kam mám umiestniť fotku od kamery.

$$\begin{aligned} w_l &= \frac{\frac{width}{2}}{\tan\left(\frac{fovy}{2}\right)} * aspectRatio \\ h_l &= \frac{\frac{height}{2}}{\tan\left(\frac{fovy}{2}\right)} \end{aligned} \tag{4.2}$$



Obrázek 4.6: Priblíženie obrázkov

Dostal som dve čísla w_l a h_l vzdialenosti v závislosti na výške a šírke, porovnám ich a budem používať väčšie, ďalej ako l. Teraz potrebujem zistiť smer kamery ($\vec{d_C}$). Tento vektor normalizujem, tým dosiahnem, že jeho veľkosť bude jednotková $\|\vec{d_C}\| = 1$. Konečne mám všetko čo potrebujem, vypočítam bod F, do ktorého premiestnim moju fotku.

$$\begin{aligned} x_F &= x_B + l * x_{\vec{d_C}} \\ y_F &= y_B + l * y_{\vec{d_C}} \\ z_F &= z_B + l * z_{\vec{d_C}} \end{aligned} \tag{4.3}$$

Na obrázku 4.6 vidím spôsob priblíženia, kde x repezentuje polovicu výšky obrázku, α je $fovy$ a d je vzdialenosť fotky od kamery.

Kapitola 5

Výsledky práce

Táto kapitola popisuje spôsob hodnotenia: aké úlohy dostali užívatelia, čo museli spraviť, na aké otázky museli odpovedať, čo som sa dozvedel o apikácii.

5.1 Spôsob hodnotenia práce

Spätná väzba od užívateľov je veľmi dôležitou súčasťou tejto práce, je to nutné k tomu, aby som získal podrobnejšie informácie o tom, že to, čo som vytvoril, je použiteľné. Spätné väzby získam formou úlohy, ktorú musia spraviť užívatelia formou dotazníka.

Potrebujem merať intuitívnosť a efektivitu použiteľnosti programu. Tieto dve veci budem merať na dvoch rôznych zobrazeniach s rôznym počtom obrázkov.

Úloha bude, že každý užívateľ dostane konkrétnu fotku, ktorú musí nájsť v scéne medzi ostatnými. Užívateľ vždy vyskúša jeden pohľad s menším počtom obrázkov ako nezaškolený, a potom druhý pohľad, ale už ako zaškolený s väčším počtom a pokúsi sa nájsť vybranú fotku. Je pritom nutné merať čas potrebný na nájdenie fotky. Tento odmeraný čas bude slúžiť na výpočet intuitívnosti a efektívnosti medzi dvoma zobrazeniami. Cieľom je porovnať kolotoč s `atraktormi`, už existujúce riešenie s novým, a pozorovať reakcie užívateľov. Väčší rozdiel bude pravdepodobne pri hľadaní medzi viacerými obrázkami. Kolotoč neumožňuje nijako rozdeľovať fotky do kategórií, ale `atraktory` áno. Teoreticky, keď je veľké množstvo načítaných obrázkov a pozná sa približne, čo sa nachádza na obrázku, ktorý hľadáme, stačí zapnúť tie `atraktory`, ktoré popíšu tú tému. Budeme hladať auto, tak určite pomôže zapnutie atraktora, ktorý bude priťahovať fotky s autami.

Každý užívateľ po vyskúšaní programu vyplní formulár s otázkami skúmajúcimi spokojnosť, pozitívne stránky a nedostatky. Som zvedavý na to, či je jasný princíp `atraktorov` a či to považujú za použiteľnú inováciu. Čo sa najviac nepáči a opačne, ktoré veci sú tie, ktoré sa najviac ľúbia. Je problém orientovať sa v trojrozmenom prostredí, alebo či sú lepšie klasické programy? Tieto sú hlavné attributy, na ktoré sa budem spýtať "budúcich"užívateľov.

5.2 Výsledky testovania

Program vyskúšali ôsmi ludia, ktorý dostali najprv test pre nezaškoleného. Niektorí museli vyhľadávať medzi 70 fotkami, ako nezaškolení, iní dostali len 10 kusov. Dôvodom, prečo dostali iný počet na prvýkrát je, že tým som meral intuíciu. Efektivita bola odmeraná tak, že hľadali rovnakú fotku medzi rovnakým počtom fotiek, ale pomocou iného zobrazenia.

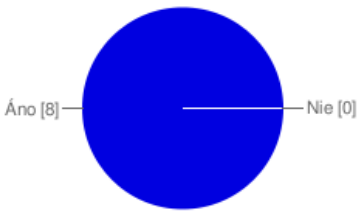
Najlepšie by bolo, keby užívatelia používali program pár dní a potom by sa opäť merali tieto údaje – dostal by som reálnejšie výsledky.

Predtým, ako sa začalo testovanie, bolo vysvetlené, na čo sú `atraktory`, ako sa dá s nimi pracovať. Užívateľ sám vyskúšal vyhľadať fotku, ktorá mu bola ukázaná. Niektorí to zvládli rýchlejšie, iným to trvalo oveľa dlhšie. Bolo vidno, že nie je jednoduché používať myš na manipulácie 3D scény. Potom, ako prebehla prvá časť testovania, každý dostal podrobnešie informácie o tom, ako používať program, ako približovať fotky alebo celú scénu, ovládanie `atraktorov`, ich premiestnenie, mohli to sami vyskúsať a pohrať sa s programom. Po školení opäť dostali úlohu, ale teraz museli vyhľadať inú fotku medzi iným počtom fotiek.

Časy, ktoré boli namerané, boli velmi rôzne, čo súvisí s tým, že testovali sa ľudia, ktorí pracujú veľa s počítačom, ale aj takí, ktorí len málo. To ale neovplyvňuje výsledky, lebo tí, ktorí spravili úlohu pre nezaškoleného rýchlejšie, boli šikovnejší aj pri úlohe pre zaškoleného. Po vyskúšaní `atraktorového` zobrazenia bola otestovaná práca so zobrazením typu kolotoč. Úlohy boli podobné, nájst fotku medzi menej a viacerými fotkami. Tu som nerozdeľoval zaškolených a nezaškolených, bolo im ukázané, ako to používať a boli odmerané časy potrebné k úlohám.

Po týchto úlohách musel každý vyplniť 6 otázok, ktoré sa týkali programu. Na dve z týchto otázok museli odpovedať s vlastnými slovami. Otázky boli nasledujúce:

Otázka č. 1 - Pochopili ste myšlienku `atraktorov`?
Na túto otázku som dostal len odpovede áno, z čoho môžem odvodiť, že pojem `atraktor` bol jasný pre každého. Užívaťelia pochopili v čom sú dobré a na čo ich môžeme používať, boli aj takí, ktorí radi by ich uvítali vo svojom prehliadači, lebo si mysleli, že by im pomohol roztriediť fotky, ktoré nevedeli rozdeliť rozumne a majú ich pohromade.

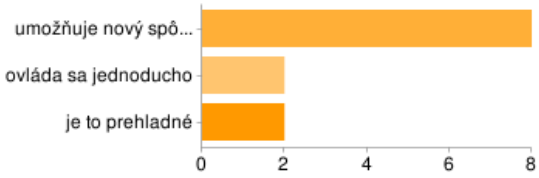


Obrázek 5.1: Pochopili ste myšlienku atraktorov?

Otázka č. 2 - Páči sa mi `atraktorové` zobrazenie, lebo:

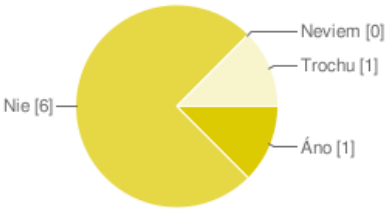
- umožňuje nový spôsob zobrazenia
- ovláda sa jednoducho
- je to prehľadné

Bola to otázka, kedy užívatelia mohli zvoliť viac možností. Každý označil možnosť, že sa mu páči, lebo umožňuje nový spôsob zobrazenia, ale odpoveď, že sa to ovláda jednoducho, naviac je to aj prehľadné, označil už len 25% testovaných. Len štvrtina si myslela, že je to jednoducho ovládateľné. Je treba ešte pracovať na tomto spôsobe zobrazovania. Tiež len štvrtina považovala to za prehľadné, čo znamená, že rozloženie nie je ani zďaleka optimálne.



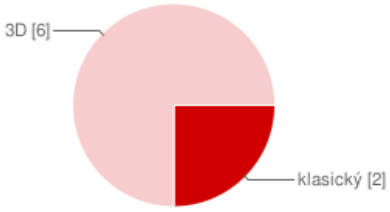
Obrázek 5.2: Páči sa mi atraktorové zobrazenie, lebo ...

Otázka č. 3 - Robí vám tažkosti orientácia v 3D?
Odpovede, ktoré som dostal na túto otázku záležia od viacerých faktorov. Tieto faktory sú hlavne vek, typ práce, aj pohlavie. Starší ľudia majú všeobecne menej kontaktu s počítačovými programami, čo má za výsledok, že nebudú ich tak šikovne ovládať. Pre ľudí pracujúcich napríklad v oblasti IT bude určite jednoduchšie orientovať sa v 3D, ako pre učiteľov základných skôl. Väčšina zvládla orietáciu, čo je celkom 75%, jeden s tým mal menšie problémy (13%) a jednému to bolo ťažšie (13%).



Obrázek 5.3: Robí vám tažkosti orientácia v 3D?

Otázka č. 4 - Ktorý spôsob ovládania vám viac vyhovuje?
Táto otázka súvisí s tým, či bola pre určitého užívateľa bezproblémová orientácia. Tí, ktorí to zvládli, budú odpovedať s menšou pravdepodobnosťou, že im vyhovuje klasický typ zobrazenia. Pod klasickým typom chápeme dvojdimenzionálne zobrazenie, teda klasické programy rozšírené v súčastnosti.



Obrázek 5.4: Ktorý spôsob ovládania vám viac vyhovuje?

Otázka č. 5 - Čo sa vám najviac nepáči?
Na otázku, čo sa najviac nepáči, boli rôzne odpovede, niektorým sa nepáčilo, že musia dvakrát kliknúť na obrázok, aby sa zväčšil. Tento problém nie je ťažké vyriešiť, ale keby sa nám obrázky zväčšili na jeden klik, mohli by sme zväčšiť aj keď to nepotrebujeme, napríklad pri

otáčaní scény. Ďalšie problémy boli: ťažká navigácia, rýchlosť otáčania, ťažká orientácia. Rýchlosť otáčania závisí hlavne na citlivosti myši, ak máme nastavenú myš na veľmi citlivú, tak nám to bude oveľa rýchlejšie otáčať.

Ťažkú orientáciu je možné zlepšiť tak, že navrhneme jednoduchšiu a prehľadnejšiu scénu, aby to nerobilo problém pre niektorých užívateľov. Vytvoriť takú scénu, ktorá bude dobrá pre každého je teoreticky nemožné. Vždy tam nájdeme veci, ktoré sú pre niekoho dobré, ale inému zasa nevyhovujú.

Jeden z užívateľov prišiel na to, že v diaľke nie je možné rozpoznať, ktorý `atraktor` je ktorý. Majú inú farbu, ale to nestačí na to, aby boli jednoznačne rozpoznateľné.

Posledná vec je, že pri priblížení v niektorých zobrazeniach je zle vypočítaná výsledná pozícia obrázku, kam sa má presunúť. Táto chyba sa objavuje len vtedy, keď kamera nie je v bode nula, na ose z. Dôvodom toho je chybný matematický výpočet. Táto chyba bola nakoniec odstránená.

Otázka č. 6 - Čo sa vám najviac páči?
Druhá otázka, na čo museli užívateľia odpovedať s vlastnými slovami bola, že čo sa im najviac páči. Mnohí písali, že sa im páči hlavne to, že je to trojdimenzionálne. Nový štýlový spôsob zobrazovania a priestorové zobrazenie vzbudzuje príjemný pocit pri prehliadaní obrázkov. Niektorým sa páčila myšlienka celého programu, spôsob hľadania fotiek, lebo je to takto oveľa zaujímavejšie, zábavnejšie ako v klasických prehliadačoch. V porovnávaní s klasickými programami tu sú animované efekty, čo sa každému ľúbilo.

Musím si uvedomiť, že tieto dve zobrazenia, ktoré som porovnával sú istým spôsobom neporovnávateľné, lebo `atraktory` umožňujú výber podľa zadaných kritérií, ale kolotoč neobsahuje žiadne prostiedky na to, aby umožnil nejakým spôsobom skupinovať fotky. U prvého sme schopní vybrať kategórie medzi ktorými hľadáme, ale u druhého budeme vždy hľadať medzi všetkými obrázkami.

Chcel som vyškúsať testy aj so zväčšenými obrázkami. Situácia sa však nezlepšila, vyzeralo to tak, akoby obrázky boli len zväčšené, totiž boli príliš veľké oproti `atraktorom`. Tento spôsob nie je dobrý na testovanie, pridal do scény zase chaos, ktorý som sa snažil eliminovať voči druhému navrhnutému spôsobu pri zobrazení `atraktorov`. K tomu, aby pomer plochy prádznych miest a obrázkov bol menší, bolo by treba iným spôsobom navhrhnúť scénu a usporiadať obrázky okolo `atraktorov` rozumnejšie.

Rovnica na výpočet intuitivity:

$$I = \left(\frac{A_{Neza10}}{A_{Za10}} + \frac{B_{Neza10}}{B_{Za10}} + \frac{A_{Neza70}}{A_{Za70}} + \frac{B_{Neza70}}{B_{Za70}}\right)/N \tag{5.1}$$

$$\begin{aligned} I &= \left(\frac{36.25}{31.05} + \frac{17.3}{20.1} + \frac{27.2}{31.55} + \frac{49.8}{38.1}\right)/4 \\ &= (1.17 + 0.86 + 0.86 + 1.3)/4 \\ &= 1.05 \end{aligned} \tag{5.2}$$

Výsledok 5.2 ukazuje, že použitie programu je intuitívne. Čím viac sa bude približovať výsledok k jednotke, tým viac bude môj program intuitívnejší. Intuitívnosť bola vypočítaná tak, že som porovnával zobrazenie typu kolotoč s `atraktorovým`. Čísla, ktoré som dostal odmeraním nezaškolených užívateľov, boli vydelené s číslami zaškolených. Mal sme dva

obrázky (obrázok A a B) a dve sady obrázkov, jednu menšiu (10 kusov) a jednu väčšiu (70 kusov). Celkom boli vypočítané ďalšie štyri čísla a z nich som musel ešte vypočítať priemer.

Rovnica na výpočet efektivity:

$$E = \left(\frac{A_{I10}}{A_{II10}} + \frac{B_{I10}}{B_{II10}} + \frac{A_{I70}}{A_{II70}} + \frac{B_{I70}}{B_{II70}}\right)/N \tag{5.3}$$

$$\begin{aligned} E &= \left(\frac{31.05}{15.175} + \frac{20.1}{9.525} + \frac{31.55}{17.6} + \frac{38.1}{24.375}\right)/4 \\ &= (2.04 + 2.11 + 1.79 + 1.56)/4 \\ &= 1.875 \end{aligned} \tag{5.4}$$

Porovnanie `atraktorového` zobrazenia s kolotočom dáva výsledok, že kolotoč je o 87.5% efektívnejší 5.4. Tento údaj ale nemôžem považovať za všeobecne platný. Závisí to od toho, akí ľudia testovali môj program. Ľudia z oblasti IT budú určite rýchlejší pri testoch, ako ľudia, ktorí s počítačom nepracujú tak veľa. Ďalší faktor,ktorý môže ovplyvňovať tieto výsledky, je počet obrázkov, medzi ktorými treba hľadať.Keď budeme mať tisíce obrázkov, kolotoč nebude ten najvhodnejší spôsob na to, aby sa tam jednoducho našli fotky, ale keď sa zvolí zobrazenie s `atraktormi`, tak už je možnosť rozdeliť fotky a určite bude jednoduchšie hľadanie.

$$E_{10} = \left(\frac{31.05}{15.175} + \frac{20.1}{9.525}\right)/2 = (2.04 + 2.11)/2 = 2.075 \tag{5.5}$$

$$E_{70} = \left(\frac{31.55}{17.6} + \frac{38.1}{24.375}\right)/2 = (1.79 + 1.56)/2 = 1.675 \tag{5.6}$$

Rozdelím efektivitu na dve časti podľa počtu fotiek a budem skúmať závislosť. Prvý údaj bude porovnanie dvoch zobrazení s menším počtom a druhý s väčším počtom. Efektivita u prvého je 2.075 a druhého 1.675. Ako vidieť, kolotoč je efektívnejší aj s menším aj s väčším počtom, ale efektivita klesá, keď sa pridá viac fotiek. Z toho vyplýva, že ak sa bude zvyšovať počet fotiek, po nejakej hranici kolotoč prestane byť použiteľným a bude lepší druhý typ, kde je možnosť výberu. Ďalšie údaje je možné nájsť v tabuľkách C.1 a C.2.

Kapitola 6

Práce do budúcna

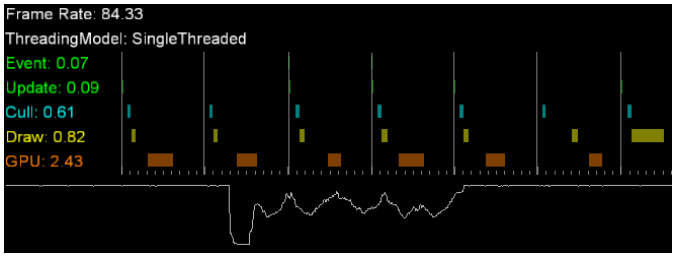
Táto kapitola popisuje aké plány mám do budúcna, ako je možné vylepšiť knihovňu, optimalizovať výpočty, ktoré zaťažujú procesor, plne využívať podporu grafickej karty a matematické výpočty a pamäť namiesto RAM. Návrhy iných zobrazení, rozmiestnenie fotiek s `atraktormi` a vylepšienie grafického užívateľského rozhrania. Vylepšenie `atraktorov` tak, aby vedeli sami sa naučiť čo sa nachádza na obrázkoch, tým by nám uľahčili prácu a celý program bude sa správať inteligentnejšie. Zavedenie viac interaktivity do nášho programu, vylepšenie grafických efektov.

6.1 Optimalizácie

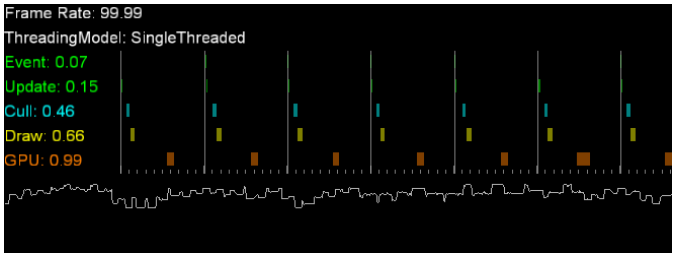
Optimalizáciu budem potrebovať nielen u matematických výpočtoch, ale aj u načítaní obrázkov. Nasledujúce obrázky ukazujú priebeh načítania fotiek a zaťaženie procesoru. Na druhom obrázku 6.1(a) vidieť, že načítanie trvalo dlhšie, čo je spôsobené tým, že ešte neexistovali miniatúry k fotkám, a program ich vytvoril, aby sa nemuselo umiestniť do operačnej pamäte obrovské množstvo.

Z načítaných fotiek je vytvorená textura a mapovaná na špeciálne uzly. Keď potrebujem originálnu fotku, napríklad keď to mám priblížené, tak stačí vymeniť miniatúru na pôvodnú. Tu je problém, že načítané obrázky obsadia veľký priestor v RAM. V `OSG` je možné explicitne nastaviť, aby ich nedržal v pamäti, ale z nejakých dôvodov to nefunguje, alebo funguje zle. Po zväčšení viacerých fotiek sa začne zaberať viac a viac miesta. Po výmene na miniatúry, teda po zmenšení tieto fotky opustia pamäť korektne. Z týchto dôvodov nie je možné načítať originálne obrázky, naviac aj zbytočné, keď ich ani nepotrebujem. Riešením, ako načítať veľké fotky, by bolo použitie LOD (Level Of Detaiľ), čím by sa zariadilo to, že keď uzol je ďaleko a nie je vidieť detaily, obsahoval by miniatúru, a ako sa približuje, by ju postupne vymenil na originálny, ktorý už obsahuje všetky detaily. Ďalšie optimalizácie bude treba urobiť pri výpočte pozície fotiek, ktorá závisí od pozícií `atraktorov`. Aby sa situácia trošku zkomplikovala, každý obrázok má inú váhu oproti `atraktorom`. To znamená, že `atraktory` budú ťahať tie fotky silnejšie, ktoré majú vyššie váhy príslušnosti. K tomu ešte treba pripočítať, že aj `atraktory` môžu mať nastavenú silu.

Výpočty sú teraz implentované na strane procesora a tieto výpočty sú časovo veľmi náročné, preto bolo by lepšie používať grafickú kartu, ktorá je efektívnejšia a rýchlejšia na aritmetické operácie, ako centrálna jednotka. Najprv sa vypočíta stred fotky pomocou `atraktorov`, potom sa musí znova prepočítať už s váhami a nakoniec ešte silami attraktov a je vypočítaná pozícia jednej fotky. Ten výpočet prebehne pri generácii každého snímku.



(a) existujú

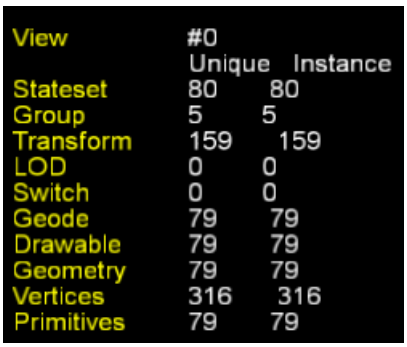


(b) neexistujú

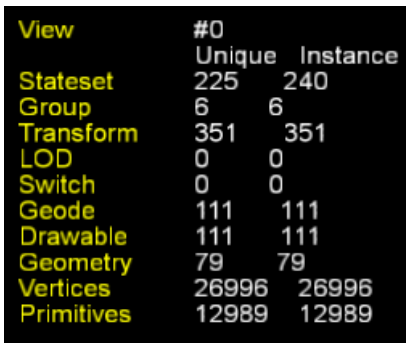
Obrázek 6.1: Štatistiky načítania obrázkov

Bude 100 obrázkov a 100 snímkov za sekundu, tak treba vypočítať 100 * 100 = 10000 pozícií. A ešte som nerozoberal, že jeden obrázok môže patriť nielen k jednému `atraktoru`. Každý pridaný `atraktor` k fotke zvýši výpočet trikrát. Aby sa nemuseli na každý snímok vypočítať všetky pozície, môže sa uložiť pozícia `atraktorov` a porovnať s novými a prepočítavať len to, čo sa zmenilo, tým sa ušetrí veľa času. Musí sa potom dávať pozor aj na silu `atraktorov`, lebo tie sú meniteľné užívateľom.

Na optimalizáciu treba dávať tiež pozor. Porovnám dve zobrazenia, jedno obsahuje `atraktory` a druhé nie. Atraktory sú reprezentované formou farebnej guľe. Táto guľa je zložená z obrovského množstva primitív a keď budem mať ich viac, nebude to optimálne riešenie. Ako vidieť na obrázku 6.2(a) je neporovnavateľne menší počet používaných primitív,ako na obrázku 6.2(b).



(a) kolotoč



(b) atraktor

Obrázek 6.2: Štatistiky

Uzly `atraktorov` obsahujú v sebe meno kategórie, z ktorej priťahujú fotky. Ak sú tieto uzly ďaleko od užívateľa, text nie je čitateľný, jednoducho nie je možné rozpoznať ich. Jednou možnosťou je, že by sa to urobilo podobne ako u obrázkov: pod myšou sa zväčší text. Tým ešte problém nie je úplne vyriešený, lebo musí sa prejsť cez všetky uzly, kým sa nenájde tá, ktorú sme hľadali.

6.2 Vylepšenie

Grafické užívateľské rozhranie je napísané v knihovni `Qt`. Všetky nastavenia by sa mohli zabudovať do `OSG`, hlavne nastavenie `atraktorov`. Vypnutie a zapnutie `atraktora` sa môže prerobiť tak, že stav bude meniteľný kliknutím na uzol, a rolovaním s myšou bude nastaviteľná sila ťahania. Vymyslieť rôzne vylepšenie nie je problém, zatiaľ je to v počiatočnom stave. Nastaviť silu `atraktorov` pomocou spinboxov nie je najvhodnejší spôsob, musí sa veľa rolovať pri nastavení.

Veľa z dnešných prehliadačov má zabudovanú podporu na dotykový display, tieto sú hlavne používané na mobilných zariadeniach. Užívatelia môžu rýchlo a jednoducho s prstom ovládať program. Tvári sa to prirodzenejšie, lebo vyzerá to podobne, ako by sa pracovalo s ozajstnými fotkami. Môžu sa chytiť, roztiahnuť, odhodiť nabok. Pridanie takej podpory z prehliadača by spravil zábavnejší program a vylepšila by sa tým aj ovládateľnosť.

Myš nie je vhodná na ovládanie trojdimenzionálneho prostredia, ako riešenie sa môže brať do úvahy *spaceball*, napríklad *SpacePilot* od firmy *3Dconnection*, obrázok 6.2. Tento nástroj bol vymyslený na podobné veci, aké sú tu potrebné, dá sa s ním hýbať do 12 smerov. Otázka je, či by užívatelia boli ochotní kúpiť ho kvôli programu. Inžinieri ho používajú pri práci v CAD programoch (*AutoCAD*, *Inventor*), ale funguje to aj s *Adobe Photoshop*, či *Google Maps*.

Obrázek 6.3: SpacePilot PRO

V budúcnosti sa určite nevyhneme vylepšeniu implementovaných zobrazení. Tieto slúžia len na prezentácu, keby sme to chceli predať ako produkt, ťažko by sa našiel zákazník, ktorému by to vyhovelo v súčasnom stave. V prípade, že je okolo `atraktora` veľa fotiek, navzájom sa prekrývajú, treba doriešiť tento problém. Stalo sa to aj pri testoch, že užívateľ hľadal fotku, ale nepodarilo sa mu nájsť, lebo bola neviditeľmá. Ďalej, musí sa zmenšiť pomer prázdnych plôch oproti využitým, kde sa nachádzajú fotky, tým sa dosiahne, že

budeme vidieť viac fotiek namiesto čierneho pozadia. Navhrnuté zobrazenia sú dobrým základom na budúce experimenty.

Váhy priťahovania a sily `atraktorov` sú počítané pomocou lineárnych funkcií, ak sa tieto vymenia, napríklad na logaritmické, vypočítané pozície sa budú chovať celkom inak. Tým sa dosiahne, že pri menších váhach obrázky budú priťahované len menej, keď budú mať väčšie váhy, tak oveľa silnejšie, stačí ak preštudujeme priebeh logaritmickej funkcie.

Kapitola 7

Záver

Výsledkom tejto práce je program, ktorý prezentuje použitie navrhnutej knihovne s podporou štyroch zobrazení. V predchádzajúcich kapitolách bol popísaný nárvh a implementácia riešenia. Výsledná aplikácia je schopná zobrazovať fotky v 3D s rôznymi zobrazeniami, medzi ktorými užívateľ môže prepínať. Túto aplikáciu som potreboval otestovať na užívateľoch a diskutovať o nedostatkoch návrhu, implementácie a vymyslieť si určité vylepšenie.

Zhodnotenie výslednej aplikácie nie je dokonalé, aplikáciu vyskúšalo len málo užívateľov, čo nestačí na odvodenie presných faktov. Tieto údaje stačia mi na to, aby som prišiel na dôležitejšie chyby. Scény sú postavené zo základných uzlov, ktoré poskytuje moja knihovňa. Pri implementácii neboli používané pokročilejšie vlastnosti `QpenSceneGraph`. Implementované boli len jednoduchšie scény, do ktorých bola pridaná menšia interaktivita v 3D.

Vývoj tejto aplikácie išiel len veľmi pomaly a to kvôli `QpenSceneGraph`. Nie sú dostupné kvalitné materiály pre začiatočníkov. Na internete treba hľadať na fórach. Je dostupná jedna kniha, ktorá rozoberá najpodstatnejšie veci v `OSG`: správu dát v pamäti, dôležité časti knihovne, stavbu scény, atď. Autor knihy predpokladá, že čitateľ bol zoznámený so základnými vlastnosťmi `OpenGL`. Táto práca bola veľmi dobrá na to, aby som získal poznatky zo sveta počítačovej grafiky a vyskúšal svoje schopnosti.

Aplikácia bola otestovaná na platforme Linux (*ArchLinux*) 32 i 64 bitový, na rôznych počítačoch. S počítačmi, ktoré majú grafikú kartu ATI a používajú oficiálny ovládač (chybná 3D podpora), môžu byť problémy, doporučujem open source ovládač. U grafických kartách NVIDIA tieto problémy sa neobjavili.

Možné pokračovanie tejto práce obsahuje predchádzajúca kapitola. Na prvom mieste sú optimalizačné práce, ako napríklad zavedenie shaderov a vytvorenie efektívnejších rozhraní pre užívateľov.

Implementáciu programu som si užíval a rád by som pokračoval v práci.

Literatura

[1] Balázs Iván József, Csonka Ferenc, Pécsy Gábor. *Java 2.* [b.m.]: ELTE TTK Hallgatói alapítvány, Budapest, 2001. ISBN 963-463-485-0.

[2] Bursuc, G. *Get Your Photo Collection Organized and Optimized* [online]. 2008 [cit. 2010-05-2]. Dostupné na: <http://www.softpedia.com/reviews/windows/Pictomio-Review-97984.shtml>.

[3] Daniel Solin. *Qt Programming in 24 Hours (Teach Yourself In 24 Hours).* [b.m.]: Sams, 2000. ISBN 978-06-723-1869-6.

[4] Ionut Ilascu. *3D Image Viewer* [online]. 2008 [cit. 2010-05-2]. Dostupné na: <http://www.softpedia.com/reviews/windows/VisionsReview-83444.shtml>.

[5] István Szentadrási. *Grafická aplikace v Pythonu s použitím OpenGL.* Bakalářská práce. Brno: FIT VUT v Brně, 2009.

[6] Jackie Neider and Tom Davis and Mason Woo. *The Official Guide to Learning OpenGL, Release 1.* [b.m.]: Addison-Wesley Publishing Company, 1994. ISBN 0-201-63274-8.

[7] Jeff Molofee (Nehe). *Nehe Productions: OpenGL Lesson #04 Rotation* [online]. Dostupné na: <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=04>.

[8] Jeff Molofee (Nehe) and Dimitrios Christopoulos. *Nehe Productions: OpenGL Lesson #30 Collision Detection* [online]. Dostupné na: <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=30>.

[9] Jeff Molofee (Nehe) and Lionel Brits (ssetelgeuse). *Nehe Productions: OpenGL Lesson #10 Loading And Moving Through A 3D World* [online]. Dostupné na: <http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=10>.

[10] Jesse Liberty. *Naučte se C++ za 21 dní.* [b.m.]: Computer Press, a.s., Brno, 2002. ISBN 80-7226-774-4.

[11] Paul Marth. *OpenGL röviden.* [b.m.]: Kiskapu Kiadó, 2007. ISBN 978-96-396-3725-2.

[12] Paul Marth. *OpenSceneGraph Quick Start Guide* [online]. [b.m.]: Computer Graphics Systems Development Corporation, Mountain View, California, 2007. Dostupné na: <http://www.lulu.com/items/volume_51/767000/767629/3/print/OSGQSG.pdf>.

[13] Robert C. Martin. *Čistý kód (Návrhové vzory, refaktorování, testování a další techniky agilního programování).* [b.m.]: Computer Press, a.s., Brno, 2009. ISBN 978-80-251-2285-3.

[14] Rudolf Pecinovský. *Návrhové vzory.* [b.m.]: Computer Press, a.s., Brno, 2007. ISBN 978-80-251-1582-4.

[15] Seth Rosenblatt. *Cooliris for Firefox 3* [online]. 2008 [cit. 2010-05-2]. Dostupné na: <`http://download.cnet.com/Cooliris-for-Firefox-3/3000-11745_4-10870217.html`>.

[16] Steve Riley. *Scenegraphs and Openscenegraph for 3D Software Development* [online [cit. 2010-05-2]]. Dostupné na: <`http://eureka3d.com/blog/?p=18`>.

[17] Zsolt Czompál. *Programovatelné shadery v OpenSceneGraph.* Bakalářská práce. Brno: FIT VUT v Brně, 2009.

Příloha A

Obsah DVD

- *browser* – Súbory k programu
  - *bin* – spustiteľný program po preložení
    - *fonts* – fonty k textom v programu
    - *browser* – spustiteľný program 32 bit Linux
  - *obj* – priečinok pre dočasné súbory (*.o)
  - *tmp* – priečinok pre dočasné súbory (*.cpp)
  - *scr* – screenshoty spravené od začiatku práce
  - *src* – zdrojové súbory
- *doc* – Dokumentácia
  - *tex* – TEX súbory
  - `technicka_sprava.pdf` – Technická správa
  - `tests.odt` – výsledky testov vo formáte OpenOffice.org Writer
- *lib* – Používané knihovne
  - *osg* – zdrojové súbory k OpenSceneGraph
  - *qt* – zdrojové súbory a príklady ku Qt
- *pic* – Obrázky
  - *data1.csv* – malá sada fotiek
  - *data2.csv* – veľká sada fotiek
- *video* – Videá
  - *vid1.mpg* – kolotoč a stena (klasické)
  - *vid2.mpg* – guľa a kolotoč s atraktormi (nové)
- `README`

Příloha B

Manual

Program je protrebné preložit pred spustením, `qmake` nám vyrobí `Makefile` v priečinku *src*:

```
qmake browser.pro
```

Teraz už môžeme preložit s príkazom:

```
make
```

Po preložení sa nám objaví v priečinku *bin* spustiteľný binárny súbor *browser*.
Na ovládanie programu je možné používať myš a klávesové skratky.

Klávesové skratky:

Ctrl + I Posunie atraktorov - zapnutie, vypnutie

Ctrl + C Otvoriť CSV súbor

Ctrl + D Otvoriť priečinok

Ctrl + Q Vypnúť program

S Informácie o grafickom hardware - FPS, graf (zátaž), ďalšie informácie

Space Východisková pozícia

B Backface culling

H Help

T Textúry - vypnutie, zapnutie

W Prepnutie medzi režimy: polygon fill, wire frame, points

1 Ovládač kolotoč

2 Ovládač atraktor

3 Ovládač stena

Myš:
Ľavé tlačítko slúží na otáčanie, pravé na priblíženie a stredné na posúvanie. Tieto funkcie záležia na ovládačov.

Príloha C

Výsledky testov

Tabulka C.1: Výsledky v sekundách - atraktor

	Test A (Obrázok č. 1)				**Test A (Obrázok č. 2)**			
	Zaškolený		**Nezaškolený**		**Zaškolený**		**Nezaškolený**	
Počet fotiek	**10**	**70**	**10**	**70**	**10**	**70**	**10**	**70**
1			55.1			54.3		
2				20.5	13.6			
3	9.0							24.6
4		27.7					18.3	
5			22.4			21.9		
6				33.9	26.6			
7	53.1							75.0
8		35.4					16.4	
Spolu	61.1	63.1	77.5	54.4	40.2	76.2	34.6	99.6
Priemer	31.05	31.55	36.25	27.2	20.1	38.1	17.3	49.8
Smerodatná odchlyka	22.27	3.85	16.54	6.7	6.5	16.2	0.9	25.2

Tabulka C.2: Výsledky v sekudách - kolotoč

	Test A (Obrázok č. 1)		**Test A (Obrázok č. 2)**	
Počet fotiek	**10**	**70**	**10**	**70**
1	9.9			18.8
2		22	7.9	
3	38			7.7
4		10.3	2.9	
5	9.5			27.7
6		29.5	19.7	
7	3.3			43.3
8		8.6	7.6	
Spolu	60.7	70.4	38.1	97.5
Priemer	15.175	17.6	9.525	24.375
Smerodatná odchlyka	13.44	11.39	6.2	14.92