Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Vizualizácia informácií v obohatenej realite

Dokumentácia k inžinierskemu dielu

Tím č. 5 - ARVis

Bc. Duško Dogandžić
Bc. Dávid Durčák
Bc. Ján Handzuš
Bc. Patrik Hlaváč
Bc. Marek Jakab
Bc. Matej Marcoňák
Bc. Daniel Soós
Bc. Martina Trégerová

Vedúci projektu: Ing. Peter Kapec, PhD.
Predmet: Tímový projekt II
Kontakt: teamtp05@gmail.com
Akademický rok: 20213/2014, letný semester

Obsah

1 Úvod

Tento dokument zachytáva technickú dokumentáciu k projektu Vizualizácia informácií v obohatenej realite v rámci predmetu Tímový projekt na Fakulte informatiky a informačných technológií na Slovenskej technickej univerzite v Bratislave. Dokument je rozdelený na dve hlavné časti - práca na technickej dokumentácii v zimnom, resp. v letnom semestri.

2 Zimný semester

2.1 Šprint 1

2.1.1 Analýza potrebných knižníc pre prácu na projekte

2.1.1.1 OpenSceneGraph

OpenSceneGraph je opensource 3D grafické API pre vývojárov. Slúži na podporu aplikácií pre virtuálnu realitu, vizualizáciu, modelovanie, obohatenú realitu, hry a simuláciu vozidiel. OSG má multiplatformovú podporu, ktorá je vhodná pre zariadenia ako telefóny a tablety, cez stolové počítače až po generovanie 3D obrazov veľkého rozlíšenia.

Keďže OSG je písané v štandardnom C++ s openGL, aplikácie môžu byť spustené na rôznych opečaných systémoch: Microsoft Windows, Mac OS X, Linux, IRIX, Solaris alebo FreeBSD. Od verzie 3.0.0 OSG podporuje mobilné platformy Android a iOS. V súčasnosti je vydaná verzia 3.2.0 stable. Jeho architektúra pozostáva z troch častí:

- **OpenSceneGraph** knižnica - ktorá je ďalej rozdelená na moduly pre manipuláciu, tvorbu, správu pamäti, knižníc pre správu vlákien, osgUtil s optimalizáciou renderovania a transformácie scény do OpenGL. Obsahuje tiež pripravené prvky pre prácu s databázou a správu dát. Je podporovaná väčšina bežných 2D/3D formátov (*jpg/gif/png/tiff/bmp iwo/obj/3ds/x*).
- **OsgViewer** - pre správu pohľadov a prostredia kamery.
- **NodeKits** - množina riešení pre základné situácie a grafických algoritmov, ako napr. *osgTerrain*, ktorý slúži na vykreslenie terénu, *osgTex*, ktorý poskytuje rozšírenú správu fontov alebo *osgFX* pre špeciálne efekty a postprocessing.

2.1.1.2 OpenCV

OpenCV (Open source Computer Vision) je multiplatformová knižnica počítačového videnia a spracovania obrazu.[1] Je napísaná v jazyku C/C++/Python/Java a podporuje operačné systémy Windows, Android OS, Linux, iOS a Mac OS.

[1] Zdroj: http://opencv.org/, dostupné z webu v novembri 2013.

Knižnica bola vytvorená v roku 1999 spoločnosťou Intel a kládla veľký dôraz na efektivitu a spracovanie v reálnom čase. V súčasnosti má viac ako 2500 optimalizovaných algoritmov a približne 6 miliónov stiahnutí. Je jednou z najviac používaných knižníc v oblasti počítačového videnia. Knižnica nesie značku open-source BSD library, čo znamená, že umožňuje voľne šíriť obsah, využívať kód pre komerčné účely bez zverejňovania zdrojového kódu.

V balíku OpenCV knižnice sa nachádza niekoľko zdieľaných a statických knižníc rozdelených podľa modulov:

- **Core** – kompaktný modul definujúci základné štruktúry a funkcie používané inými modulmi.
- **Imgproc** – modul pre spracovanie obrazu. Nachádzajú sa tu lineárne a nelineárne obrazové filtre, geometrické obrazové transformácie, histogramy a podobne.
- **Video** – modul pre analýzu videa, ktorý zahŕňa odhadovanie pohybu, odstraňovanie pozadia alebo sledovanie objektov.
- **Calib3D** - modul pre základné geometrické algoritmy, kalibráciu kamery, odhad pozície objektu a rekonštrukciu 3D obrazu.
- **Features2D** - modul pre detektory lokálnych príkazov, deskriptory a párovanie deskriptorov.
- **Objdetect** -modul pre detekciu objektov a inštancií predefinovaných objektov (tvár, oči, ľudia, autá).
- **Highgui** - modul pre jednoduché rozhranie na zachytávanie videa, obrazu.
- **Gpu** - modul pre grafickú akceleráciu algoritmov z rôznych vyššie spomínaných OpenCV modulov.

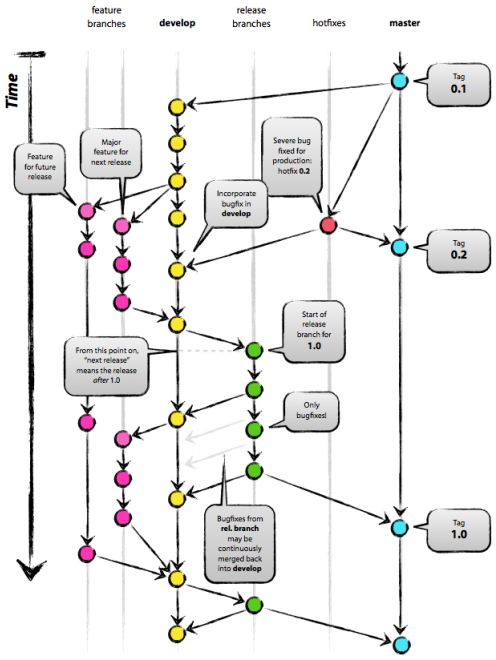
Pre potreby aplikácie vizualizácie dát v obohatenej realite je možné využiť openCV knižnicu pre rozpoznávanie tváre, ktorej poloha sa môže využiť pri vykresľovaní dát. Algoritmus pre rozpoznávanie tváre prebieha pomocou detekcie Haarových príznakov. Súčasťou openCV sú k dispozícii aj natrénované príznaky pre detekciu tváre.

Ďalšou zaujímavou funkcionalitu OpenCV je rozpoznávanie planárnych objektov metódou lokálnych deskriptorov. Algoritmus najprv vyhľadá kľúčové body na obrázku a opíše ich okolie. Detekcia kľúčových bodov prebieha pomocou rôznych detektorov, ktoré sa zameriavajú na význačné časti na obrázku. Tieto body sú význačné tým, že je

možné ich ľahko nájsť aj pri rôznych transformáciách obrázku. Príkladom môžu byť hrany. Okolie daných kľúčových bodov následne opíšeme vektorom čísel - deskriptory. Rovnaký proces sa aplikuje na obraz scény. Následné sa párujú deskriptory a zistuje, či sa daný objekt nachádza na scéne. Metóda lokálnych deskriptorov je pomerne robustná a máme k dispozícii viaceré deskriptory ktoré sú navyše invariantné voči rotáciám, škálam, ale aj rôznym svetelným podmienkam. V knižnici OpenCV sa nachádzajú deskriptory ako SIFT, SURF, BRIEF, ORB.

2.1.2 Analýza GitFlow

GitFlow je model o stratégiách vetvenia a verziách v rámci verziovacieho systému Git navrhnutý Vincentom Driessenom, na riadenie a spoluprácu vo väčších projektoch. Tento model definuje striktný model vetiev okolo manažmentu zverejnenia projektu. Tento model sa snaží o definovanie špecifických úloh jednotlivých osôb vo vetvách a zároveň, ako by mali čo najefektívnejšie spolupracovať. Pričom veľký dôraz sa kladie na využívanie *features* vetvy, ako izolovaný vývoj s možnosťami experimentovania a ako prostriedok na efektívnejšiu spoluprácu.



Obr. 1: *GitFlow*[2]

[2]Zdroj: http://nvie.com/posts/a-successful-git-branching-model/, dostupné z webu v novembri 2013.

V modeli sú zadefinované dve hlavné vetvy, ktoré existujú počas celého vývoja produktu, ako to vidíme na obrázku 1:

- *master* – zobrazuje typickú hlavnú vetvu, ktorá by mala obsahovať produkt, ktorého stav je pripravený na produkciu,
- *develop* – vetva, o ktorú sa opiera vývoj, v nej sa zobrazujú vývojárske features pre ďalšie vydanie projektu.

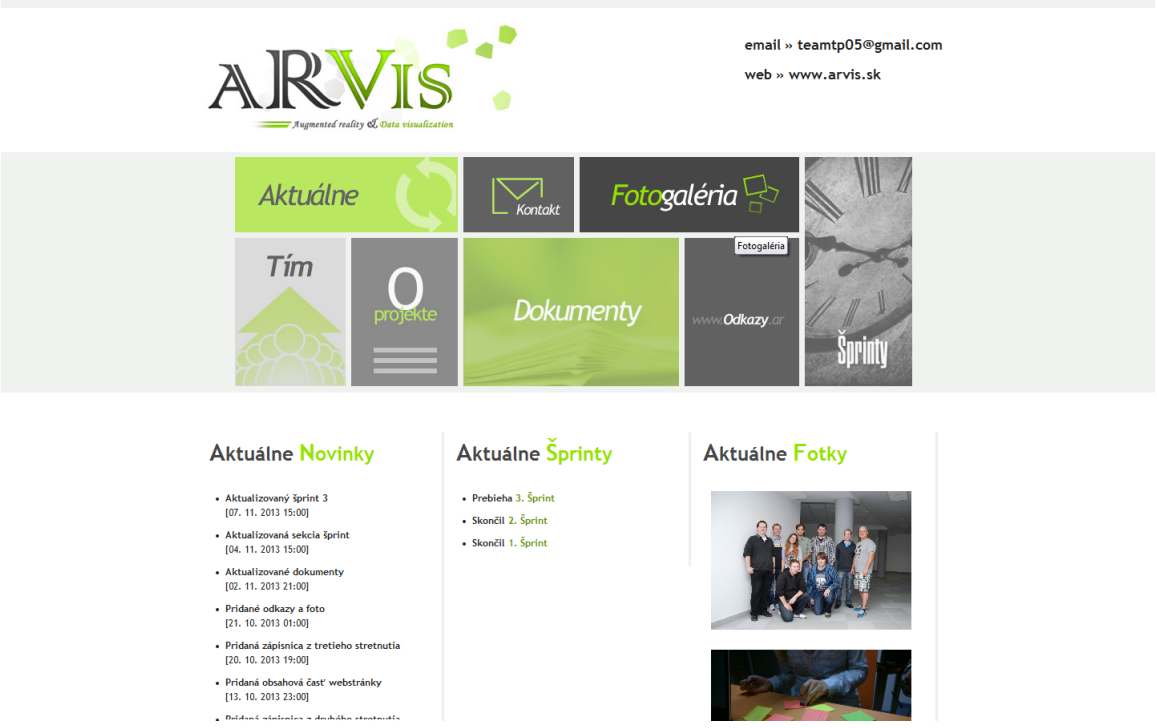
V rámci vývoja sa priamo nevyvíja na tieto dve vetvy. V master vetve máme ukážkový projekt, ktorý môžeme odoslať na produkciu a *develop* vetva slúži na integráciu jednotlivých funkcionalít z vedľajších vetiev. Keď *develop* vetva dosiahne stabilný bod, tak všetky zmeny sa presunú do vetvy master a začína sa nové produkčné obdobie.

Vedľajšie vetvy slúžia na pomoc pri vývoji z hľadiska pomoci pri sledovaní nových fukcionalít (features), prípravu produktu na produkcií, asistenciu pri oprave chýb (obr. 1). Na rozdiel od predošlých vetiev, tieto vetvy majú obmedzenú existenciu. Základné definované vetvy sú:

- *features* – tieto vetvy sú vytvárané pre novú funkcionalitu pre nasledujúcu, poprípade budúcu produkciu. Tieto vetvy vychádzajú priamo z *develop* vetvy, a v žiadnom prípade nemôžu integrovať s *master* vetvou. Táto vetva existuje počas vývoja funkcionality a po dokončení je spojená s *develop* vetvou a zanikne.
- *Hotfix* – vetva je vytváraná pre rýchlu opravu chýb, ktoré vznikli. Neprináša žiadnu novú funkcionalitu, ale o opravu definovanej chyby. Následne sa oprava spojí s príslušnou hlavnou vetvou a zanikne.
- *Release* - táto vetva vzniká pred prípravou na produkciu. V stave, keď *develop* vetva obsahuje dostatok nových funkcií, urobí sa z nej kópia do *release* vetvy. Následne sa táto vetva pripravuje na vydanie a následne spojenie s *master* vetvou. Existencia tejto vetvy je krátka, nepridávajú sa žiadne väčšie funkcionality, čisto len príprava na produkciu.

2.1.3 Vytvorenie webstránky

Webstránka ARVis je uložená na školskom serveri, dostupná aj cez doménu www.arvis.sk. Prezentácia sa skladá zo statických HTML stránok, pričom na spojenie jednotlivých súborov sa okrajovo využíva jazyk PHP. Spolu s nástrojom pre Scrum – Redmine, používajú na zobrazenie server Apache. Na serveri je tiež úložisko dát pre zápisnice a dokumentácie alebo tiež obrázky do fotogalérie. Webstránka tímu č. 5 je jednoduchá HTML prezentácia so štýlmi CSS a prvkami JavaScriptu (knižnica jQuery). Pozostáva z dvoch obsahovo odlišných častí. Úvodná stránka s jednoduchým efektom zobrazenia navigácie umožňuje prechod na obsahové stránky. Tieto stránky sa členia podľa zamerania obsahu na niekoľko kategórií. V sekcii äktuálne"sú pravidelne dopĺňané informácie o činnosti tímu, pričom obsahujú linky do príslušných sekcií. O úlohách, zadní projektu a členoch tímu sú vytvorené samostatné sekcie, rovnako ako sekcia "šprinty", ktorá obsahuje údaje zo zápisníc, pričom umožňuje lepší celkový prehľad.



Obr. 2: *Zobrazenie stránky 'Aktuálne'*

2.2 Šprint 2

2.2.1 Analýza osgART

Je to vývojárska knižnica pre C++ platformy, ktorá sa používa pri vytváraní obohatenej reality. Kombinuje 3D grafickú knižnicu s knižnicami pre sledovanie používateľa a jeho okolia. Zabezpečuje rôzne stupne funkcionality ako napr. vysokoúrovnňovú integráciu vstupu z videa, rozoznávanie priestoru, fotometrické rozoznávanie.

Podporuje Linux, Mac OS, aj Windows. Pre Windows je nutná podpora Visual Studia, pri Mac OS X Code. Pri Linuxe stačí štandardný g++ toolchain. Ďalšie informácie ohľadom softvérových požiadaviek sa nachádzajú na oficiálnej stránke knižnice[3].

Pred samotnou inštaláciou je potrebné mať nainštalovaný *OpenSceneGraph* a *ARToolKit*. Požiadavka na hardvér je mať funkčnú kameru. Pri inštalácii treba pri všetkých platformách stiahnuť zdrojové súbory a tieto buildnúť pomocou *CMake* listov. Ďalšie zdroje riešia špecifické problémy pre rôzne platformy, resp. obsahujú konkrétne inštalačné skripty nielen na osgART.

- `http://augmentedrealityworks.blogspot.sk/2011/07/installing-artoolkit-in-windows-using.html`
- `http://tech.enekochan.com/2012/06/10/install-osgart-2-0-rc3-with-openscenegraph-2-8-3-with-collada-support-in-ubuntu-12-04/`
- `https://github.com/enekochan/installation-scripts`

2.2.2 Analýza Kinectu

Kinect je periférnym zariadením na detegovanie pohybu. Microsoft pre toto zariadenie poskytuje SDK na svojich platformách Windows 7 a Windows 8. Pre tento projekt je dôležité, že SDK podporuje vývoj v C++ a ako integračné vývojové prostredie mu primárne slúži Visual Studio.

Ako alternatíva k SDK existuje open-source projekt *OpenKinect*[4], ktorý logicky má podporu na Linux aj OS X platformách, na rozdiel od oficiálneho SDK. Konkrétne sa sústreďujú momentálne na projekt *libfreenect*[5]. Výhody oficiálneho SDK:

[3]Zdroj: http://www.artoolworks.com/support/library/Requirements_to_use_osgART#Required_software, dostupné z webu v novembri 2013.

[4]Zdroj: http://openkinect.org/wiki/Main_Page, dostupné z webu v novembri 2013.

[5]Zdroj: https://github.com/OpenKinect/libfreenect, dostupné z webu v novembri 2013.

- prehľadná dokumentácia a kvalitná webová komunita – aj OpenKinect má napr. mailing list, ale chýba niečo „oficiálne“,
- hardvér je tiež produktom Microsoftu, takže oficiálne SDK s tým korešponduje – open source komunita musí ísť cestou spätného inžinierstva.

Nevýhody oficiálneho SDK:

- podpora len pre Windows,
- podpora pre menší počet programovacích jazykov (v tomto projekte to žiadny efekt nemá, nakoľko sa programuje v C++).

OpenNI je ďalším open-source SDK pre senzory, ale táto spoločnosť vyvíja SDK pre vlastný senzor. V súčasnoti už ale OpenNI podporuje aj Kinect. Takisto ako OpenKinect, aj OpenNI má repozitár na GitHub. Existuje projekt, ktorý používa aj kinectovské MS SDK nad OpenNI[6]. V tomto riešení sa ešte používala staršia verzia OpenNI, ale v súčasnosti je kompatibilný s MS produktom aj OpenNI 2.0.[7] Nad iným OS ako Windows tento bridge nefunguje, to znamená, že spolupráca MS SDK s OpenNI je rovnako platformovo limitovaná ako samotný MS SDK. Spolupráca OpenNI a OpenSceneGraph sa rieši na diskusnom fóre osg vo väčšej miere ako libfreenect na tomto fóre, aj keď hlavne sa jedná o osgAnimation. OpenCV je taktiež vysoko kompatibilné s OpenNI. OpenNI má značnú výhodu oproti libfreenect v tom, že podporuje vysokoúrovňové funkcionality ako fragmentácia scény, sledovanie človeka a rozpoznávanie gest. Naopak, funkcionality bližšie k hardvéru ako akcelerátor, LED alebo ovládanie motora poskytuje len libfreenect (od OpenKinect). K OpenNI je aj middleware *NITE*, ktorý poskytuje algoritmy, ktoré využívajú informácie od hardvéru. Podobne ako OpenNI, aj NITE je multiplatformové.

2.2.3 Fixácia kompilačných chýb v predchádzajúcom riešení

Pre kompiláciu 3DVisual na platforme Mac OSX bolo potrebne odstrániť niekoľko kompilačných chýb:

- zlé zadefinovanie ternárneho ()?():() operátora v súbore src/Viewer/CoreGraph.cpp. Oprava si žiadala správne použitie zátvoriek.

[6]Zdroj: https://code.google.com/p/kinect-mssdk-openni-bridge/, dostupné z webu v novembri 2013.

[7]Zdroj: http://stackoverflow.com/questions/14491963/how-to-setup-openni-2-0-with-opencv-for-a-kinect-project, dostupné z webu v novembri 2013.

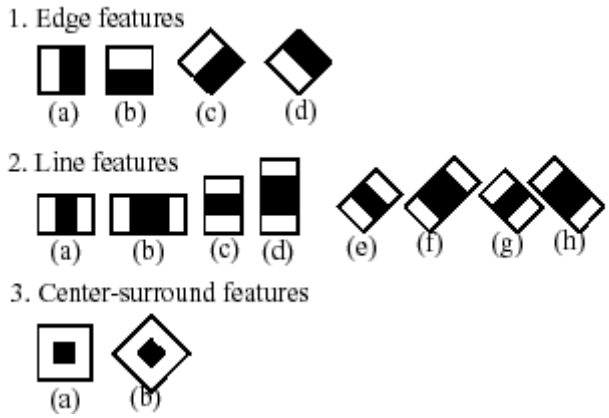
- Kompilátor mal problém s použitím knižníc Qt/qstring.h a Qt/qstringlist.h. Pre vyriešenie tohto problému bol pridaný ifdef blok, ktorý v prípade kompilácie na platformách apple používa knižnice qstring.h a qstringlist.h namiesto hore spomynaných. Zmena bola nutná v súboroch:
  - include/QOSG/CoreWindow.h
  - include/QOSG/MessageWindows.h
  - include/Util/ApplicationConfig.h

2.2.4 Úprava 3DVisual pre novšie verzie knižníc

Softvér 3DVisual využíva niekoľko starších verzií knižníc. Analyzovalo sa niekoľko novších verzií týchto knižníc, ktoré sú voľne dostupné, a ktoré by bolo možné aktualizovať v našom projekte. Identifikoval som OpenSceneGraph knižnicu, ktorú je možné s malými úpravami aktualizovať z verzie 3.0.1 na 3.2.0.

2.2.5 Rozpoznanie tváre

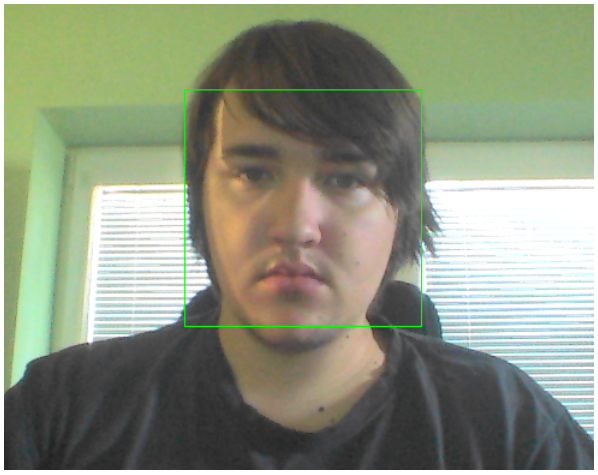
Pre funkciu rozpoznania tváre ako prvú funkcionalitu sme sa rozhodli použiť knižnicu openCV. V tejto knižnici sa nachádza súbor obsahujúci haar-like príznaky pre detekciu rôznych častí tela. Pre naše potreby sme využili súbor *haarcascade_frontalface_alt.xml* s naučenými príznakmi pre detekciu tváre.



Obr. 3: *Haar-like príznaky (features)*

Úspešne sme implementovali algoritmus rozpoznávania tváre ako externý program a v ďalšom kroku plánujeme jeho import do celkového programu. Pre každý obrázok,

ktorý nám vráti kamera, vykonáme detekciu všetkých tvárí na obrázku a vykreslíme okolo nej zelený štvorček. Vzhľadom na povahu našej aplikácie neskôr eliminujeme detekciu všetkých tvárí na obrázku a sústredíme sa len na jednu detegovanú tvár.



Obr. 4: *Rozpoznanie tváre*

2.2.6 Aktualizácia 3DVisual o funkcionalitu z iných vetiev na GitHube

2.2.6.1 Analýza 3DVisual

Pri analýze programu 3DVisual sme identifikovali, že tento program používalo viacero študentov ako základ ich bakalárskej alebo diplomovej práce. Bohužiaľ väčšina pridanej funkcionality je roztrúsená v rôznych vetvách, a preto je vhodné preskúmať tieto jednotlivé funkcionality a následne ich zaradiť do programu. O tento proces bola už snaha aj v minulosti, no však nepodarilo sa pridať všetky funkcionality, ktoré sú dostupné.

2.2.6.2 Návrh

Na základe 3DVisual sme identifikovali existujúce vetvy, ktoré vznikli. Následne je potrebné identifikovať množstvo zmien, ktoré boli vykonané a presne, o ktorú funkcionalitu bol program rozšírený. Podmienkou pre rozšírenie programu o novú funkcionalitu je, aby tento proces netrval dlho z dôvodu následného vývoja.

2.2.6.3 Implementácia

Z existujúcich riešení sme si vybrali tie, ktorých funkcionalita je pre rozšírenie softvéru zaujímavá a má prínos do budúcnosti. Následne sme vykonali nasledujúce kroky:

- identifikovali problémy v jednotlivých funkcionalitách:
  - používanie zbytočných header súborov,
  - nekorektné pridávanie nových metód do abstraktných tried,
  - zavedenie inej metodiky pri písaní premenných
  - nekorektná úprava už existujúcich metód.
- Spojenie neproblémových súborov.
- Riešenie problémových súborov pri spájaní:
  - rozhodovanie o redundantných funkciách,
  - nekorektných premenných,
  - zbytočných includoch,
  - tvorba chýbajúcich metód,
  - oprava chýbajúcich referencií na premenné.
- Oprava kompilačných chýb:
  - oprava nekompletných metód,
  - opravy týkajúce sa multiplatformovému behu programu.
- Testovanie pridanej funkcionality na základe existujúcich dokumentácií.

2.2.6.4 Zhodnotenie aktualizácie

Výsledná aktualizácia obsahuje novú funkcionalitu, ktorú sme otestovali a zistili obmedzenia, ktorými táto nová funkcionalita disponuje. Následne po skončení testovania sme túto verziu označili ako produkčnú a následne sa od nej budú implementovať naše ďalšie funkcionality.

2.3 Šprint 3

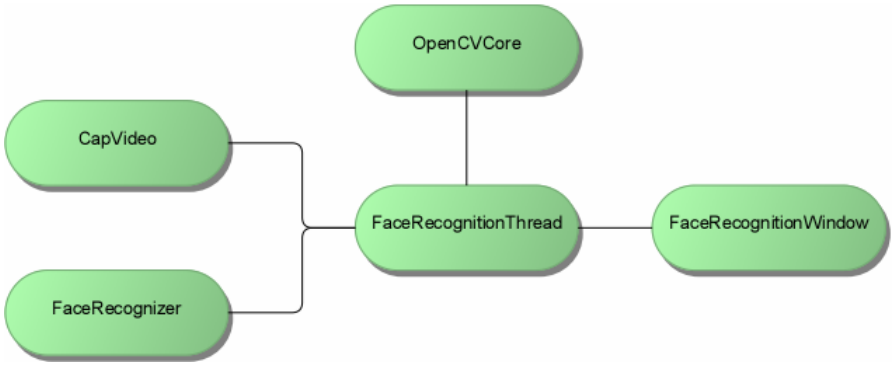
2.3.1 Import OpenCV

Pre import openCV funkcionality do multiplatformového programu sme najprv museli nastaviť potrebné knižnice. V CMakeLists bolo pre to potrebné nastaviť:

- # Find OpenCV
- FIND_PACKAGE(OpenCV)
- IF (OpenCV_FOUND)
- INCLUDE_DIRECTORIES($OpenCV_INCLUDE_DIRS)
- ENDIF()

Následne sme mohli pridať zdrojové súbory pre rozpoznávanie tváre. Rozhodli sme sa vytvoriť triedu OpenCVCore, ktorá bude vytvárať všetky funkcionality spojené priamo s knižnicou openCV.

Funkcionality, ktoré sme implementovali v podobe ďalších tried boli načítanie a práca s videom z kamery (trieda CapVideo) a samotné rozpoznanie tváre (trieda FaceRecognizer). Pre ukážku funkcionality sme vytvorili okno QDialog (trieda FaceRecognitionWindow), ktorá komunikuje s vláknom, ktorý obsahuje samotný algoritmus pre rozpoznanie (trieda FaceRecognitionThread). Prepojenie jednotlivých vidíme na obrázku 5.



Obr. 5: *Triedy a ich architektúra*

Do hlavného okna sme pridali tlačidlo pre spustenie danej funkcionality. Po jeho stlačení sa vyvolá okno so záznamom videa, kde sa vyznačí detegovaná tvár. Ďalším postupom pri tejto funkcionalite bude prepojenie kamery scény s pozíciou detegovanej tváre. Celková oblasť tváre je veľká a preto sa pri určovaní pozície zameriame na pozíciu očí.

2.3.2 Modularita v projekte

2.3.2.1 Plugin a moduly

Plugin je kód, ktorý sa môže stať častou programu, pričom je následne riadený programom. V C++ o moduloch vravíme ako o zdieľaných knižniciach, ktoré sa môžu dynamicky načítavať programom. Zdieľané knižnice, ktoré sú založene na základe pluginu.[8] Majú nasledujúce vlastnosti:

- načítanie pluginu pomocou interfacu,
- dynamicky prístup s pluginom,
- odpojenie pluginu počas behu programu.

Moduly sú mechanizmy na zapuzdrenie implementácie knižníc. Od klasického princípu prekladu sa rozlišujú v tom, že všetko sa zadefinuje v jednom mieste. Tieto súbory s týmito dátami sa nazývajú ïnterface súboryä nahradzujú používateľom písané hlavičkové súbory. Generovanie týchto súborov však vyžaduje rôzne prístupy k rozdielnym situáciám. Z toho hľadiska sa v kóde využívajú rôzne obmedzenia a rozšírenia, ktoré pri tomto procese pomáhajú.[9] Výhody sú:

- zrýchlenie build času,
- dynamický framework knižníc.

Návrhom riešenia v súčasnom projekte je identifikácia častí programu, ktoré by bolo možné týmito jednotlivými spôsobmi upraviť.

2.3.2.2 Implementácia

Počas prieskumného prototypovania sa vytvorili prípravné súbory a na základe nich sa rozhodlo, že v rámci projektu by bolo skôr lepšie použitie druhej metódy. Pri práci na projekte sa však zistilo, že celková komplexita už existujúceho riešenia pre tento proces by bola zbytočne zdĺhavá. Pri tomto procese sa však následne identifikovali miesta programu, na ktoré by bola vhodná podobná optimalizácia.

[8] Zdroj: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3347.pdf, dostupné z webu v novembri 2013.

[9] Zdroj: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n2015.pdf, dostupné z webu v novembri 2013.

2.3.3 Úprava cmake súborov pre submodul Libnoise

2.3.3.1 Opis úlohy a jej analýza

S dôvodu pomalej kompilácie sme sa rozhodli vyčleniť externé súbory patriace k modulu Libnoise, ktoré sú kompilované v spolu s projektom do samostatného submodulu vo forme externej knižnice. Vykonané zmeny však nefungovali na platforme Windows pri použití MSVC kompilátora.

Úlohou bola identifikácia príčiny problému. Príčinou bolo, že MSVC kompilátor defaultne nevytvára import lib knižnice.

2.3.3.2 Riešenie

Boli identifikované 2 možné spôsoby riešenia. Prvým je pridanie makier a značiek pre kompilátor do hlavičkových súborov. Tento spôsob je príliš náročný, preto sa pristúpilo k 2. spôsobu, ktorým bolo pridanie .rc a .def súbor pre procesu kompilácie knižnice, čo si vyžadovalo dodatočnú úpravu cmake súborov.

2.3.3.3 Zhodnotenie

Úpravou cmake súborov sa dospelo k odstráneniu problému s linkovanim submodulu Libnoise. Zmeny boli úspešne otestované na Linux aj Win platforme.

2.3.4 Globálny pohľad na program

3DVisual je aplikácia pre zobrazovanie grafov v 3D priestore. Má implementovaný algoritmus pre rozmiestnenie grafov do priestore a aplikovanie viacerých ohraničení pre tento graf. Je vyvíjaná multiplatformovo s použitím frameworku Qt v.4.8 a pre zobrazovanie je použitá knižnica OpenSceneGraph s ďalšími pomocnými knižnicami.

2.3.4.1 Súčasný stav aplikácie

Aplikácia v súčasnom stave umožňuje:

- Otvárať a vykresľovať súčasne len 1 graf zo súboru GraphML, RSF a GXL.
- Vyberať a ukladať grafy aj s ich rozmiestnením do databázy.
- Aplikácia umožňuje pracovať s nasledujúcimi typmi uzlov a hrán:
  - Klasické uzly – pohybujú sa v závislosti od ostatných uzlov pôsobením príťažlivých a odpudivých síl.

  - Fixné uzly – ako klasické uzly, no nemôžu samovoľne nemenia svoju polohu. Ak s nejakými vybranými klasickými uzlami pohneme, automaticky sa zmenia na fixné.
  - Meta uzly – pomocné uzly, ktoré pridáva používateľ pre iné uzly. Majú fixovanú pozíciu a pôsobia silami na zvolené uzly.
  - Multi hrany - viacero hrán s pomocnými uzlami spájajúce 2 uzly.
  - Hyper hrany - hrana, ktorá spája viac ako 2 uzly, pridáva sa pomocný uzol.
  - Hyper uzly - uzly, ktoré majú vhniezdené ďalšie uzly a hrany a predstavujú tak vnorené grafy.

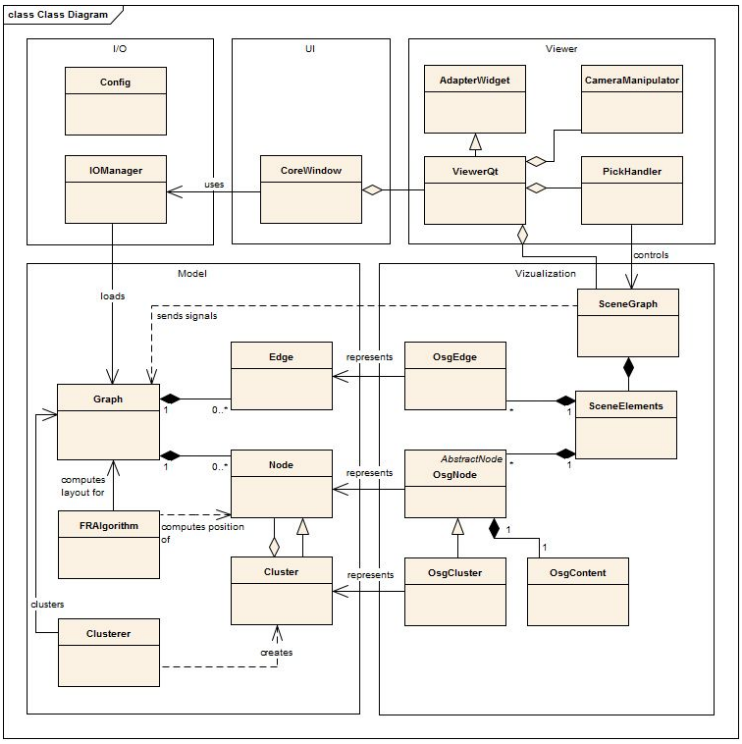
- Pohybovať grafom a aj kamerou v 3D ohraničenom priestore okolo zobrazeného grafu.
- Vo vykreslenom grafe umožňuje výber jedného alebo viacerých uzlov a hrán, ktoré je možné ďalej presúvať, meniť im farby.
- Možnosť zvoliť uzol alebo skupinu uzlov ako stred grafu okolo ktorého sa otáča kamera
- Vyberať medzi 2 typmi pozadia.
- Možnosť zapnúť a vypnúť zobrazovanie popisov pre uzly.
- Možnosť zastavenia a spustenia vykresľovania grafu a nastavenie veľkosti síl pôsobiacich medzi uzlami.
- Možnosť aplikovať viaceré druhy ohraničení na zvolenú skupinu uzlov.
- Podporuje kolaboratívnu prácu s grafom.

2.3.4.2 Súčasná architektúra systému

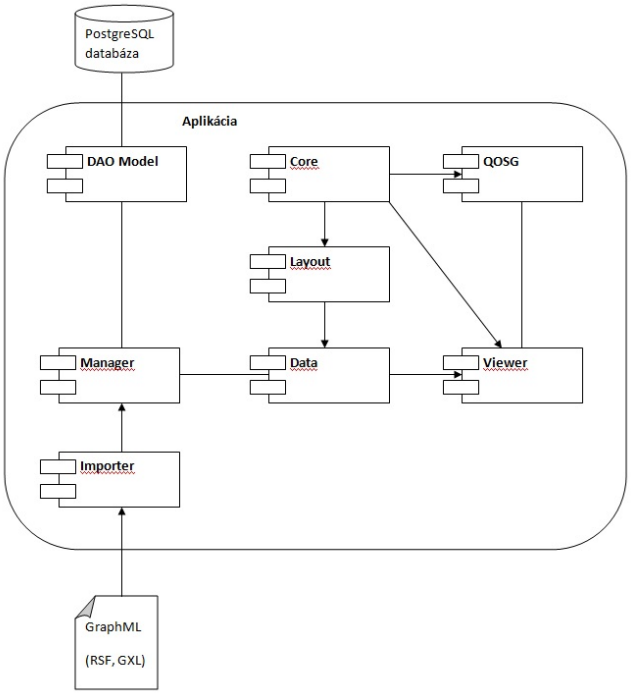
V aktuálnom stave je aplikácia rozdelená do niekoľkých modulov, podľa ich funkcie. Obr. 7 zobrazuje pôvodnú architektúru systému. Pre lepšie vysvetlenie obr. 6 zobrazuje podobnú štruktúru avšak s pohľadu hlavných tried.

- **Core** - jadro systému, inicializuje základné časti systému(CoreGraph, CoreWindow, FRAlgorithm).

- **Data** - dátový modul pre opis štruktúry grafu, obsahujúci triedy reprezentujúce jednotlivé prvky grafu (graph, node, edge, type, layout, ...).
- **Importer** - modul pre parsovanie vstupných vstupných súborov vo formátoch GraphML, RSF a GXL.
- **Layout** - modul pre rozmiestnenie uzlov v 3D priestore. Obsahuje implementáciu layoutovacieho algoritmu, a taktiež triedy pre pridávanie ohraničení rozmiestnenia.
- **Manager** - modul pre prácu s grafom.
- **Math** - model pre rozšírenie práce s kamerou.
- **Model** - modul, ktorý poskytuje rozhranie medzi systémom a databázou a mapuje jednotlivé objekty na tabuľky. Poskytuje funkcie pre pripojenie sa k databáze, výber z a ukladanie grafov do databázy. Umožňuje ukladanie uzlov aj s ich atribútmi a viacero rozmiestnení pre 1 graf.
- **Network** - modul pre podporu kolaboratívnej práce nad grafom. Poskytuje klient/server funkcionalitu.
- **Noise** - modul pre vytvorenie generovaného 3D priestoru pre pozadie.
- **OsgBrowser** - modul zabezpečuje mapovanie udalostí pre jednotlivé klávesy a pohyb myši medzi Qt a OpenSceneGraph rozhraním a vizualizáciu načítaných grafov.
- **QOSG** - modul zabezpečujúci grafické používateľské rozhranie. Obsahuje hlavné okno a ďalšie pomocné okna a widgety.
- **Util** - modul zabezpečujúci konfiguráciu nastavení aplikácie a funkcie pre vyčistenie pamäte.
- **Viewer** - modul pre pohyb v 3D priestore a prácu s kamerou. Zabezpečuje tiež prípravu grafu jeho pre zobrazenie a vytvorenie 3D kocky pre pozadie.



Obr. 6: *Rámcový diagram tried pôvodného systému*



Obr. 7: *Architektúra pôvodného systému*

2.3.5 osgART prieskumné prototypovanie

Prieskumným prototypovaním osgART sme zistili, že jedna z jeho kľúčových častí, ARtoolkit nie je možné nainštalovať na platformu Mac OS X. Taktiež vznikol problém pri inštalácií na iných platformách, kde sa podarilo nainštalovať ARtoolkit, avšak osgART nepracoval správne. Hlavným dôvodom je, že vývoj voľne dostupných verziií ARtoolkit a osgART sa zastavil pred niekoľkými rokmi. Preto je potrebné nájsť iné a novšie riešenie, ktoré bude voľne dostupné. Ako jedno z vhodných riešení pre obohatenú realitu sa ponúka projekt ArUco.

2.4 Šprint 4

2.4.1 Pohyb kamery podľa tváre

V štvrtom šprinte bolo vykonané rozhodnutie využiť implementovanú detekciu tváre pre pohyb kamery vo virtuálnom priestore. Podúlahmi tejto úlohy sú:

- výpočet pozície očí,
- komunikácia (kamera-thread pre výpočet pozície očí),
- pohyb kamery na základe údajov z pozície očí.

2.4.1.1 Výpočet pozície očí

Z predošlej implementovanej funkcionality sme mali hotovú časť detekcie tváre, kde sa okolo detegovanej tváre vykreslil zelený štvorec. Údaje o tomto štvorci sme využili pri výpočte približnej pozície očí.

Vychádzali sme z predpokladu, že oči sa nachádzajú približne v 2/3 výšky tváre. Pozíciu očí na X súradnici sme zjednodušili na stred tváre.

V tejto úlohe boli zároveň upravené hlavičkové súbory, kde sa odstránili prebytočné *include* a taktiež bolo implementované tlačidlo 'Pause' s funkcionalitou pre pozastavenie vykonávaného vlákna pre rozpoznávanie tváre.

2.4.1.2 Komunikácia (kamera-thread pre výpočet pozície očí

V nástroji Qt, v ktorom je implementovaný tento projekt, sa využívajú pre komunikáciu medzi dvomi rôznymi objektami takzvané *signaly* a *sloty*. Túto funkcionalitu sme využili aj pri splnení ďalšej úlohy, kde sme potrebovali preniesť údaje o pozície očí z objektu vytvoreného vlákna do objektu, ktorý pracuje s kamerou vo virtuálnej scéne programu. Pozíciu očí sme transformovali, aby ukazovali pozíciu očí ako percentuálnu vzdialenosť od stredu obrazu. Škála pre pozíciu očí bola teda daná intervalom $<-1,1>$ pre X aj Y súradnicu.

2.4.1.3 Pohyb kamery na základe údajov z pozície očí

Predošlé výstupy z jednotlivých úloh sa využili pri pohybe kamery vo virtuálnej scéne. Bola vytvorená slot funkcia pre objekt *CameraManipulator*, ktorá bola napojená na signal z vlákna pre výpočet pozície očí. Ďalšia funkcionalita, ako vytvorenie funkcie pre pohyb kamery a jej úprava sú popísané v úlohe 'Pridanie slotu pre otáčanie grafu podľa pozície tváre'.

2.4.2 Pridanie slotu pre otáčanie grafu podľa pozície tváre

2.4.2.1 Analýza

Pre dosiahnutie efektu otáčania grafu podľa pozície tváre je potrebné dosiahnuť aj správne natočenie virtuálnej kamery, ktorá reprezentuje pohľad používateľa na graf. Na obr. 8 je znázornené toto otočenie. Vľavo je iba jednoduché otočenie kamery. Môžeme si všimnúť že toto je principiálne ekvivalentné opačnému otočeniu samotného objektu vo virtuálnom priestore. Toto však na dosiahnutie efektu , pri ktorom obrazovka predstavuje len okno, cez ktoré sa pozeráme na objekt, ktorý je pevnou súčasťou prostredia, čiže má stálu pozíciu voči tomuto oknu, nestačí. Preto je okrem Otočenia kamery potrebná aj spätná korekcia projekcie kamery, tak ako je to zobrazené na obrázku vpravo. Preto je potrebné riešiť 2 čiastkové úlohy: výpočet novej pozície kamery podľa tváre a korekcia projekcie.

2.4.2.2 Návrh a implementácia

V pôvodnej verzii programu sa pre je pre manipuláciu s kamerou používajú metódy triedy **CameraManipulator**. Aktuálne otočenie kamery je reprezentované quaternionom $\_rotation$. Pri otáčaní kamery klasickým spôsobom je pravým tlačidlo myši je nové otočenie vypočítané prostredníctvom metódy $trackball\,()$, ktorá otáča kameru súčasne 3 smermi.

Kvôli zachovaniu pôvodnej funkcionality sme sa rozhodli pri otočení nemodifikovať priamo tento quaternion, ale pridávame nový quaternion $\_rotationHead$, ktorý bude vyjadrovať dodatočné otočenie podľa pozície tváre. Jeho otočenie je vypočítane tiež funkciou $trackball\,()$, avšak osobitne v horizontálnom a osobitne vo vertikálnom smere, aby otáčanie bolo korektné len v 2 smeroch.

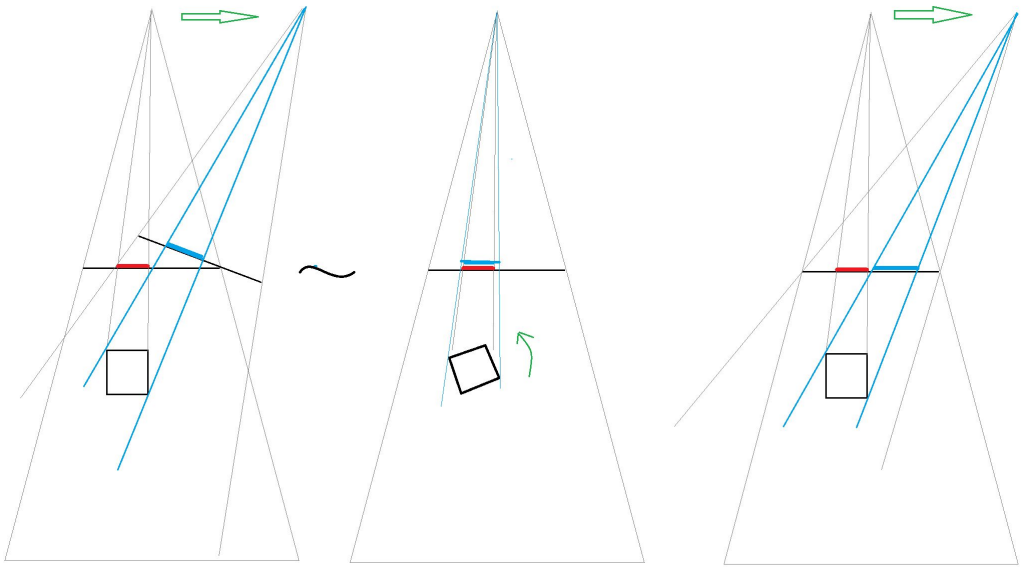
Pre korekciu projekcie je využitá metóda $getProjectionMatrixAsFrustum\,()$, ktorá umožňuje definovať akýkoľvek tvar projekcie. V tejto fáze je použitá jednoduchá korekcia, kde kamera snímajúca tvár používateľa leží priamo v strede zobrazovacej plochy, sníma priamo v smere jej normály. Preto je potrebné pridať ešte možnosť nastavenia korekcie pre pozíciu reálnej kamery.

Tieto zmeny sú implementované prostredníctvom slotu, ktorý bol pridaný do tejto triedy, a ktorý zachytáva signál zasielajúci x a y koordináty tváre a jej vzdialenosť.

2.4.2.3 Zhodnotenie

Súčasné otáčanie kamerou a dodatočná korekcia projekcie sa po testovaní ukázali ako dobrá kombinácia pre dosiahnutie efektu, že objekt je v stálej pozícií voči projekčnej

rovine. V tejto fáze nie je zatiaľ úplne korektne zapracované otáčanie a korekcia projekcie podľa vzdialenosti tváre od kamery. Obraz je zatiaľ mestami trhaný, čo je však spôsobené na strane detekcie tváre.



Obr. 8: *Otočenie kamery podľa pozície tváre*

2.4.3 Vytváranie modulov

Predošlou analýzou sme zistili, že v programe 3DSoftviz existujú 3 časti, ktoré sú z hľadiska funkcionality čiastočne nezávislé. Prvá časť je *NetWork*, ktorá obsahuje funkcionalitu na komunikáciu medzi viacerými klientami programu a kolaboratívnu prácu, ďalšia je *Database*, ktorá obsahuje funkcionalitu spojenú s ukladaním údajov a posledná identifikovaná časť bola *Importer*, ktorý slúži na načítavanie grafu. V tejto časti úlohy sa zameriavame na vytvorenie modulov z jednej tejto časti a následne refaktoring programu za účelom zníženia prepojenia jednotlivých časti a odstránenia zbytočných *include* príkazov v programe.

2.4.3.1 Implementácia

Prvým krokom v postupe vytvorenia modulov bola úprava *Cmakelists.txt* súboru, v ktorom sa upravilo jednotlivé načítanie *.cpp* súborov. Následne sme vybrali časť programu *Importer*, v ktorom sme identifikovali potrebné súbory a závislosti tejto časti. Vytvorenie modulu bolo robené na systéme Linux. V prípade vytvárania modulu v systéme Windows,

bude ešte potrebná úprava súborov vzhľadom na problémy kompilátoru Microsoft Visual Studio.

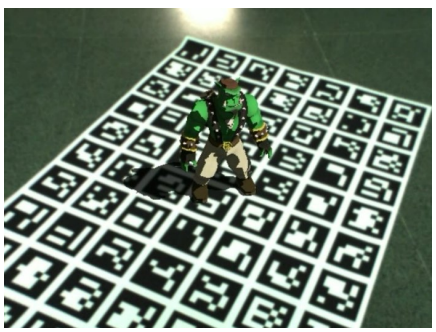
Ďalšou časťou úlohy bola úprava jednotlivých súborov a ich závislosti medzi nimi z hľadiska použitých *include* v nich. V jednotlivých súboroch boli odstránené nepoužívané *include* príkazy a skúmaná závislosť medzi ďalšími časťami.

2.4.4 ArUco - prieskumné prototypovanie

Cieľom bolo vyskúšať nastroj pre prácu s obohatenou realitou, ktorého vývoj stále pokračuje a vyhovuje našim požiadavkam.

2.4.4.1 Analýza

ArUco je softvér, ktorý využíva OpenCV, OpenGL a OGRE pre prácu s obohatenou realitou.[10] Toto docieli detegovaním značky, ktorá predstavuje Hammingov kód. Po nájdení značky v priestore vytvorí štruktúru, v ktorej je okrem iného aj translačná matica, ktorá predstavuje polohu a natočenie značky, na ktorú môžeme následne vykresliť virtuálny objekt. Deteguje 1024 značiek, ktoré môže rozlišovať po jednom alebo ako tabuľku viacerých značiek. Použitie tabuľky prináša výhody najmä, keď detegovanie jednej značky zlyháva ako napr. pri rýchlom pohybe kamery, slabom osvetlení a rôznych druhov absorpcie. Taktiež, čim viac bodov z tabuľky ArUco deteguje, tým presnejšie je schopný vykonať všetky potrebne výpočty.



Obr. 9: *Virtuálny objekt vykreslený pomocou ArUco na tabuľku značiek*
[10]

Identifikované podúlohy:

- inštalácia - úspešne ukončená, ArUco bolo otestované, príkladý sa skompilujú a spustia,

[10]Zdroj http://www.uco.es/investiga/grupos/ava/node/26, dostupné z webu v decembri 2013.

- práca s príkladmi - kednotlivé príklady boli spustené a prezreté aj na úrovni zdrojových kódov.
- práca so značkami - pri práci so značkami sme pracovali s predošlými príkladmi a na základe týchto sme vytvorili jednoduchý program, ktorý načíta značku a na základe tej hýbe objektom v scéne.

Po úspešnej a jednoduchej inštalácii, prezretí príkladov, ktoré boli v rámci knižnice a pochopeniu fungovania knižnice sme dospeli k rozhodnutiu túto knižnicu využívať aj v ďalších fázach nášho projektu.

Keďže ArUco pracuje s OpenCV knižnicou, ktorá už je v systéme integrovaná kvôli riešeniu úlohy rozpoznávania pohybu tváre, nie je nutné okrem knižnice ArUco zväčšovať počet knižníc v projekte. Knižnica samotná je veľmi jednoduchá, intuitívne použiteľná a celkovo veľmi dobre využiteľná.
ArUco - matica:

- ArUco + osg + OpenCV,
- ArUco a jeho import do 3DSoftviz,
- pohyb grafu podľa značky.

Ďalšia podúloha na štvrtý šprint bolo zistenie, ako ArUco pohybuje s objektom, nájdenie kódu, kde sa vytvára matica a kde sa získava.

Tieto časti kódu boli nájdené, zistilo sa, že je tu veľmi jednoduchá spolupráca s OpenCV knižnicou a nebude problém implementácie s ohľadom na OpenCV.

Samotná implementácia do 3DSoftViz bola z časových dôvodov presunutá do piateho šprintu ako úloha, keďže sa ukázalo, že implementácia do programu a spojazdnenie pohybu grafu podľa značky bude časovo aj implementačne náročnejšie než sme očakávali.

2.4.4.2 Inštalácia na Mac OS X

Bez väčších problémov sme nainštalovali ArUco na platformu Mac OS X. Inštalácia si vyžiadala iba drobné úpravy v zdrojových kódoch, ktoré sa týkali zahrnutia OpenGL do projektu. Podrobnejšie sme sa zaoberali ukážkami, ktoré boli zahrnuté v projekte. Pre prácu s ArUco je potrebný kalibračný súbor z OpenCV, avšak túto kalibráciu sa nám nepodarilo vykonať, tak sme použili súbor, ktorý sa dal stiahnuť na stránke spolu s inými testovacími dátami. Postupne sme vyskúšali a podrobne preskúmali všetky zdrojové kódy.

2.4.5 Import OpenNI2 a NiTe2

Pre prácu s Kinect zariadením sa v rámci analýzy problematiky rozhodlo pre použitie knižníc OpenNI2[12] a NiTe2[13]. Druhá menovaná knižnica priamo podporuje detekciu človeka a sledovanie ruky, kým OpenNI2 inicializuje zariadenie a rieši tok vstupujúcich dát. Keďže nastavenie projektu pre prácu s externými knižnicami sa rieši v hlavnom *Cmakelists.txt*, bolo potrebné napísať *.cmake* súbory k týmto dvom knižniciam, aby ich program našiel, nakoľko v štandardnom prehľadávacom priečinku neexistujú tieto súbory. V týchto dvoch vytvorených súboroch sa nachádza cesta ku knižničnému súboru (*Object File Library*) a k hlavnému hlavičkovému súboru (*OpenNI2.h*, resp. *NiTe.h*). Nalinkovanie dvoch knižníc je v tomto štádiu vyriešené len pre platformu Windows, takže je potrebné ošetriť v projektových nastaveniach aktuálny operačný systém, aby sa dalo spojiť *feature* vetvu s *develop* vetvou takým spôsobom, že vývoj softvéru mohol pokračovať aj bez toho, že by bolo potrebné nájsť knižnice, resp. samotné zariadenie Kinect. Vo Windowse sa nainštalovaním Kinect for Windows MS SDK[14] bez ďalších potrebných nastavení deteguje ovládač pre toto zariadenie, takže okrem inštalácií a implementácie *.cmake* súborov nebolo potrebné vykonať iné záležitosti pre testovanie funkčnosti knižnice v projekte 3DSoftviz.

V našom projekte je nastavený $CMAKE\_MODULE\_PATH$ nielen na implicitný adresár, ale je definovaný adresár aj v našom projekte, kde sa prehľadávajú *.cmake* súbory po zadaní príkazu $FIND\_PACKAGE$. Ideálnym riešením sa javí v rámci vykonania *commit* správy pridať samotné súbory do projektu, tým pádom žiaden ďalší člen tímu nemusí sťahovať vytvorené súbory, a nepotrebuje ich ani pridávať do štandardnej cesty modulov Cmake. Je však nutné zmeniť v enviromentálnych premenných obsah premenných $CMAKE\_INCLUDE\_PATH$ a $CMAKE\_LIBRARY\_PATH$, tieto premenné sa však už modifikovali počas inštalovania programu 3DSoftviz, takže pridať cesty nie je žiadnym problémom pre ostatných členov tímu.

Vo štvrtom šprinte sa oproti pôvodným plánom nepodarilo importovať tieto zmeny do projektu, avšak testy na lokálnom repozitári boli úspešné na jednej zo starších vetiev. V budúcnosti sa očakáva pridanie tejto knižnice do spoločnej *develop* vetvy.

[12] Zdroj: https://github.com/OpenNI/OpenNI2, dostupné z webu v decembri 2013.
[13] Zdroj: http://www.openni.org/, dostupné z webu v decembri 2013.
[14] Zdroj: http://www.microsoft.com/en-us/kinectforwindows/, dostupné z webu v decembri 2013.

2.5 Šprint 5

2.5.1 Optimalizácia detekcie tváre

V piatom šprinte sa v rámci rozpoznávania tváre uskutočňuje jeho optimalizácia, nakoľko detekcia tváre potrebuje pre svoju prácu pomerne veľké množstvo výpočtovej sily. Identifikované podúlohy sú:

- výpočet vzdialenosti tváre od kamery,
- tracking tváre - určenie oblasti detekcie tváre,
- tracking tváre - prednostné rozpoznanie tváre v oblasti predošlej detekcie.

2.5.1.1 Výpočet vzdialenosti tváre od kamery

Na základe veľkosti detegovanej tváre sme do programu implementovali získanie vzdialenosti používateľa od kamery. Táto funkcionalita vznikla ako prípadná náhrada zariadenia Kinect, ktoré obsahuje v obraze informáciu o hĺbke. Princíp výpočtu vzdialenosti tváre na 2D obraze pracuje na porovnaní celkovej veľkosti tváre (vykresleného štvorca okolo tváre).

Takto vypočítaná hĺbka bola pridaná do funkcie, ktorá spolu s pozíciou očí posiela pre kameru aj informáciu o vzdialenosti.

2.5.1.2 Tracking tváre - určenie oblasti detekcie tváre

Pri detekcii tváre sme sa stretávali s problémom detekcie, keď sa na obraze nachádzalo tvárí viac. Algoritmus pre detekciu pracuje v smere zľava zhora obrázku a pre pohyb kamery potrebujeme pozíciu prvej rozpoznanej tváre, preto sa stávalo, že keď do záberu prišla druhá osoba, kamera sa posunula podľa novo detegovanej tváre.

Kvôli tomuto faktu, ale aj kvôli zníženiu záťaže sme sa rozhodli vybrať oblasť obrázku v okolí detegovanej tváre cez *Region of Interest* (RoI) knižnice OpenCV, ktorú využijeme v nasledovnej úlohe detekcie tváre.

2.5.1.3 Tracking tváre - prednostné rozpoznanie tváre v oblasti predošlej detekcie

Po tom, ako sme úspešne detegovali tvár a vybrali oblasť okolia tváre, môžeme dané súradnice vybranej oblasti RoI využiť pre ďalšiu detekciu. V nasledujúcom obraze získanom z kamery zisťujeme výskyt tváre človeka už len v danom úseku. Znižujeme tak výpočtovú náročnosť, ale aj zamedzujeme problémom, keď sa do oblasti kamery dostala

ďalšia osoba. Celý princíp je založený na fakte, že tvár človeka sa nepohybuje rýchlo a je teda veľmi pravdepodobné, že sa tvár bude vyskytovať v podobnej polohe ako na predchádzajúcom obraze z kamery. V prípade, že sa tvár nenájde, prehľadávame opäť celý obraz kamery.

Momentálne je táto optimalizácia rozpoznávania sprevádzaná problémami pri škálovaní detegovanej tváre, keďže algoritmus pre detekciu tváre pri menšom obraze v ktorom má detegovať upravuje hodnotu pre vykreslenia štvorca. Nastávajú tak pri pohybe kamery drobné skoky. Pracuje sa na odstránení tohto problému.

2.5.2 Projekcia podľa vzdialenosti

2.5.2.1 Analýza

Táto úloha nadväzuje na úlohu *pridanie slotu pre otáčanie grafu podľa pozície tváre* z minulého šprintu a konkrétne jej podúlohu *korekcia projekcie*. Úlohou bolo začať brať pri korekcií matice aj vzdialenosť tváre používateľa. Ďalším problémom bola zmena veľkosti okna, pri ktorej dochádzalo k deformácii zobrazovanej plochy.

2.5.2.2 Návrh a implementácia

Asymetrickú projekciu definuje 6 parametrov, ktoré je potrebné správne určiť. V našom riešení je korekcia projekcie nevrhnutá tak, aby pozorovací uhol pri vzdialenosti *1* približne *60* stupňov, čo bolo určené pevne stanoveným N, ktorý reprezentuje parameter projekcie $zNear$. Ďalej budeme pokračovať len vysvetlením výpočtu parametrov $left$ a $right$, pretože ostané dva sa vypočítajú analogicky. Pre lepšie pochopenie je táto situácia zobrazená na Obr. 10.

Ako vstup dostávame koordináty pozície hlavy x, y a $distance$. Následne je potrebné určiť skutočný koordinát $xReal$.

$$xReal = distance * zNear$$

Môžeme si všimnúť, že $xReal + right = zNear/2$, z čoho nakoniec dostaneme oba vzťahy pre výpočet oboch parametrov. Je potrebné si uvedomiť, že na rozdiel od $zNear$ je $xReal$ normalizovaná hodnota, preto je potrebné ich vynásobiť.

$$right = zNear/2 - zNear/2 * xReal = zNear/2 * (1 - xReal)$$

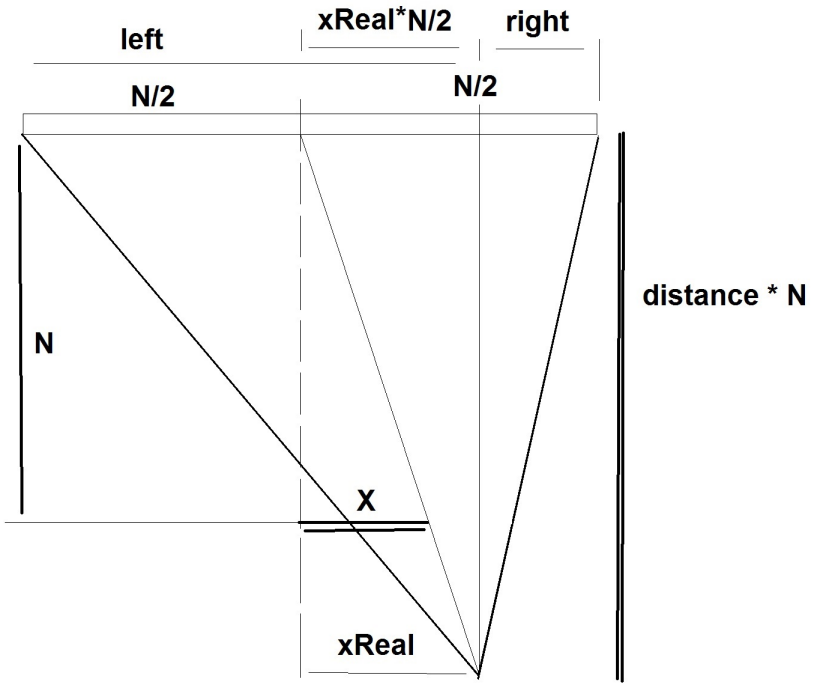
$$left = -(zNear - right) = -zNear + right$$

Ďalej si treba uvedomiť, že do nových nastaveniach dávame ako parameter pôvodnú hodnotu $zNear$, pričom sme počítali v novej projekcii s prenásobenými hodnotami vzdialenosťou, preto je tieto ďalšie parametre ešte vynásobiť dodatočne vzdialenosťou.

Ďalej na začiatku ešte získavame hodnotu $ratio$, ktorá vyjadruje pomer strán a túto tiež musíme ešte dodatočne prenásobiť horizontálne parametre, aby nedochádzalo k deformácii pri zmene pomeru strán.

2.5.2.3 Zhodnotenie

Po otestovaní sa tento spôsob korekcie projekcie správal podľa očakávaní a graf sa pri zmene pozície tváre vrátane približovania a vzďaľovania javil, akoby sa naň pozeralo cez okno.



Obr. 10: *Vysvetlenie vzdialeností a vzťahov pri projekcii*

2.5.3 Tvorba modulov

V predchádzajúcom šprinte sa nám poradilo vytvorenie modulu *Importer*. V tomto šprinte plánujeme pokračovanie práce na moduloch z hľadiska vytvárania ďalších možných modulov a úpravy už vytvoreného modulu, aby bol funkčný aj na systéme Windows.

2.5.3.1 Implementácia

Pri používaní kompilátora od systému Microsoft Visual Studio bolo potrebné na vytvorenie dynamického modulu zadefinovanie funkcií, ktoré chceme do tohto modulu začleniť. Toto dosiahneme buď použitím .def súboru, ktorý obsahuje zadefinované tieto funkcie, alebo za použitia makra *__declspec(dllexport)*, ktorým dosiahneme export dát, funkcií a tried. Pri implementácií v našom systéme je vhodnejšie použitie druhého spôsobu, vzhľadom, že funkcionalita sa v tomto module môže rozširovať.

Na systéme Linux, ktorý netrpí týmto problémom pri vytváraní dynamických modulov sa pokračovalo v testovaní ďalšieho balíku, ako napr. balík *Math* obsahuje funkcionalitu spojenú s výpočtami okolo kamery a výpočtov súvisiacich s grafmi. Ďalším balíkom na testovanie bol *Model*, ktorý obsahuje dátový model grafu. Obe tieto triedy z hľadiska ponúkanej funkcionality sú vhodné na vytvorenie samostatných modulov z nich, čo preukázalo aj testovanie. Následne prebieha implementácia v systéme Windows, ktorá je tá istá ako v prípade modulu *Importer*.

2.5.4 Zmena virtuálnej scény za reálnu

Úloha má za cieľ vymeniť virtuálnu scénu, ktorá je teraz v našom programe za reálnu scénu z kamery. Na tento účel použijeme už zakomponovaný modul OpenCV na detekciu tváre.

2.5.4.1 Analýza problému

Keďže sme sa rozhodli nepoužiť OSGart, je potrebne nájsť spôsob ako video z kamery prepojiť do OSG a nášho programu. Riešenie sa skrýva v ukážkach a v implementácii OSGart. Keďže OSG pracuje iba z objektami, treba vytvoriť objekt, na ktorý sa použije textúra obrázku z OpenCV. Táto textúra sa bude v pravidelných intervaloch aktualizovať. Treba vyriešiť niekoľko problémov:

- konvertovať obrázok z OpenCV do OSG formátu,
- naviazať pohyb virtuálnej kamery na náš objekt tak, aby napr. pri pohybe kamery nebolo vidno okraje nášho objektu,
- vyriešiť vzdialenosť objektu tak, aby v prípade pohybu grafu po scéne sa graf neschoval za náš objekt.

2.5.5 ArUco - pohyb grafu

Identifikované podúlohy:

- ArUco – import do 3DSoftViy,
- ArUco + 3DSoftViz - pohyb grafu.

Danú úlohu implementácie sme rozdelili do dvoch podúloh. V prvej je vyriešené vytvorenie *Core* triedy, cez ktorú bude ArUco komunikovať, vytvárať maticu, ktorá je aj jeho návratová hodnota.

V druhej sa už implementuje samotný pohyb grafu pomocou danej matice.

2.5.6 Statické testovanie

Po vytvorení vetvy na testovanie bolo možné riešiť túto úlohu pomocou nástroja *Cppcheck*[15], ktorý je voľne dostupný a slúži na statickú analýzu, resp. testovanie C/C++ zdrojových kódov. Rozšírenie pre QtCreator nebolo možné spustiť, ale samotný Qt projekt je možné analyzovať týmto nástrojom.

Následne sa v programe vytvoril nový projekt s hlavičkovými a zdrojovými súbormi tímového projektu. Vďaka tomuto testovaniu sa vygeneroval textový súbor, ktorý dáva za výstup chýb v programe. Cppcheck separuje typy chýb a na základe tohto sa identifikovali tie, ktoré je možné bezproblémovo opraviť. Po oprave jedného typu chýb bol program spustený úspešne. Výsledkom tejto úlohy v nasledujúcom šprinte bude vetva, kde sú vyriešené všetky chyby identifikované Cppcheck programom.

[15]Zdroj: http://cppcheck.sourceforge.net/, dostupné z webu v decembri 2013.

3 Letný semester

Táto kapitola obsahuje opis práce na projekte v letnom semestri. Štruktúra dokumentácie sa zmenila a namiesto zdokumentovaných šprintov sa tu nachádza opis v samostatných podkapitolých k vykonaným opravám oproti pôvodnému riešeniu, takisto ako opis pridaných funkcionalít, objasnenie celkového pohľadu na program. Zároveň definujeme ešte nedokončené časti projektu a výsledky testovania. V prílohách sa nachádza inštalačný a používateľský manuál k softvéru.

3.1 Vykonané opravy chýb a refaktoring

V programe chýbali ikonky k funkcionalite na obmedzenie grafu. Tieto následne boli vytvorené a zakomponované do programu.

Bola upravená funkcionalita okolo výberu grafu z priečinka. Klasické okno sa nahradilo QT náhradov okna, v ktorom bola upravená špecifická cesta k súborom.

Celkovo program prešiel optimalizáciou v rámci analytickej analýzy kódu, kde boli vyhľadané kritické miesta a opravené. Taktiež pre kompilátor GCC sme nastavili prísnejšie pravidlá ohľadom kontroly varovných hlášok. Taktiež boli tieto chyby postupne opravované.

Pokračovalo sa s optimalizáciou kódu v rámci hľadania zbytočných zahrnutí hlavičkových súborov. Následne úprava hlavičkových súborov na doprednú deklaráciu a presunutie funkcionality do cpp súborov, aby hlavičkové súbory obsahovali len deklaráciu premenných a funkcií.

```
set( FLAGS_FOR_DEBUG "-ansi -pedantic
-Wall -Wcast-align -Wcast-qual
-Wchar-subscripts -Wcomment -Wconversion -Wctor-dtor-privacy
-Wdisabled-optimization -Wextra -Wfloat-equal
-Wformat -Wformat-nonliteral -Wformat-security
-Wformat-y2k -Wformat=2 -Wimport -Winit-self
-Winline -Winvalid-pch -Wlogical-op -Wlong-long
-Wmissing-braces -Wmissing-declarations -Wmissing-field-initializers
-Wmissing-format-attribute -Wmissing-include-dirs
-Wmissing-noreturn -Wnoexcept -Wpacked
-Wparentheses -Wpointer-arith -Wredundant-decls
-Wreturn-type -Wsequence-point -Wsign-compare
-Wsign-promo -Wstack-protector -Wstrict-aliasing=2
-Wstrict-null-sentinel -Wstrict-overflow=5 -Wswitch
-Wswitch-default -Wswitch-enum -Wtrigraphs -Wundef
-Wuninitialized -Wunknown-pragmas
-Wunreachable-code -Wunsafe-loop-optimizations
-Wvariadic-macros -Wvolatile-register-var
-Wwrite-strings -Wno-unused-parameter -Wno-unused" )
```

3.1.1 Pridanie cotire

Cotire (compile time reducer) je CMake modul, ktorý urýchľuje systém "buildu"technikami ako sú prekompilované hlavičky súborov vytvorenie jednotného unit buildu. Tento modul sa používa na najvyššej vrstve - v CMakeLists.txt. V ňom sa zaradí *include(cotire)* a následne sa aplikuje na zdrojové súbory. Potrebné je taktiež nastavenie správnej závislosti medzi knižnicami, ktoré sa používajú.

```
add_executable( ${3DSOFTVIZ_NAME} ${INCL} ${SRC} )# link ${3DSOFTVIZ_NAME}_unity target
    to all libraries to which ${3DSOFTVIZ_NAME} links
set_target_properties( ${3DSOFTVIZ_NAME} PROPERTIES
COTIRE_UNITY_LINK_LIBRARIES_INIT "COPY" )
cotire( ${3DSOFTVIZ_NAME} CONFIGURATIONS Debug Release )
```

Celkové zrýchlenie systému je závislé od viacerých faktorov ako kompilátor, používaný hardware a komplexita kódu C++.

3.1.2 Úprava načítavania grafu zo suboru

Jednou z dôležitých úprav oproti pôvodnej aplikácii bola úprava načítavania z grafu. Bez úpravy pri načítavaní zo súboru dochádzalo k zbytočnej snahe tento graf hneď uložiť do databázy. To spôsobovalo predĺženie načítavania aj veľmi jednoduchých grafov o prednastavený limit 5 sekúnd na otestovanie pripojenia k databáze. Preto sme toto upravili v triede *Manager*. V pôvodnej implementácii bola v menu možnosť uložiť len layout pre graf, nakoľko tento sa uložil pri načítaní. Keďže toto uloženie sa odstránilo, tak sme pridali osobitne túto možnosť aj do menu a vytvorili pre ňu potrebné metódy, aby táto funkcionalita nebola stratená.

3.2 Súčasná architektúra aplikácie

V tejto kapitole uvádzame podrobnejší opis finálnej architektúry aplikácie. Najskôr opisujeme celkový pohľad na architektúru a v ďalších podkapitolách opisujeme nové moduly a zmeny vykonané v pôvodných moduloch. Opisujeme tiež funkcionalitu, ktorá bola takto pridaná.

3.2.1 Celkový pohľad na architektúru aplikácie

Na diagrame komponentov (Obrázok 11) je znázornená finálna architektúra aplikácie. Nevykonali sme žiadne zásadné zmeny do pôvodnej architektúry systému. Väčšina pôvodných modulov zostala z hľadiska architektúry vo svojom pôvodnom stave. Zmeny sme

vykonali len v prípade, že bolo potrebné prepojiť niektorý z nových pridaných modulov s niektorým z pôvodných modulov. Takto boli upravené moduly:

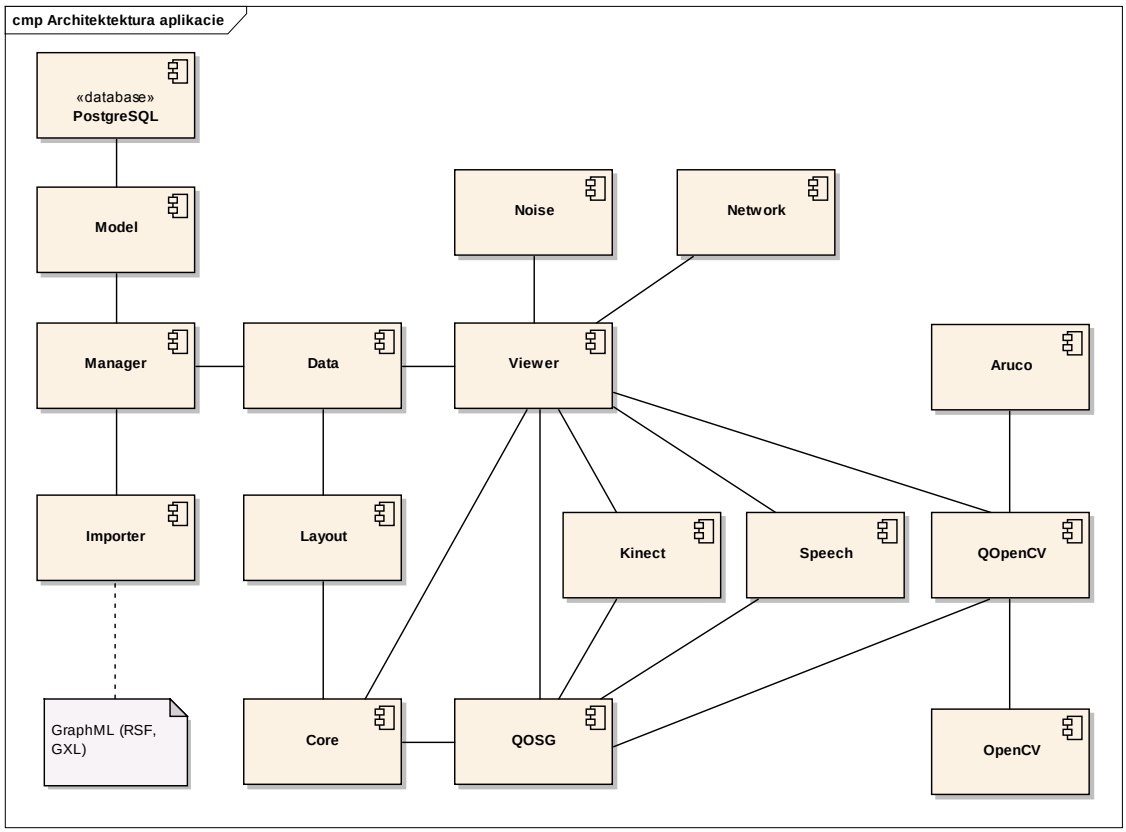
- *Viewer* - úpravy pre účely otáčania a pohybovania grafom a kamerou, označovanie uzlov a ovládanie hlasom,
- *QOSG* - zmeny v používateľskom rozhraní pre používanie novej funkcionality,
- *Manager, Model* - úprava práce s databázou pri načítavaní grafu zo súboru.

Nové moduly, ktoré boli pridané a v pôvodnej verzii sa nevyskytovali sú:

- **Kinect** - tento modul zabazpečuje ovládanie a komunikáciu so zariadením Kinect. Získava informácie z Kinectu, ktoré ďalej spracováva. Implementuje funkcionalitu pre rozpoznávanie gest, ktoré sú potom využité pre ovládanie myši alebo otáčanie a pohybovanie grafom, a tiež gest pre ďalšie ovládacie funckie.
- **Speech** - je to nový modul, ktorý implementuje funkcionalitu rozpoznávania hlasu.
- **OpenCV** - tento modul obsahuje triedy pre rozpoznávanie tváre na obraze z kamery, ďalej poskytuje funkcionalitu pre správu kamier. Zahŕňa triedu dynamicky meniteľného obrázku pre textúru v OpenSceneGraphe - použitého pre funkcionalitu video pozadia a tiež controller triedu pre správu týchto funkcionalít a ovládanie rozpoznávania značky.
- **QOpenCV** - modul obsahuje spoločné okno pre ovládanie rozpoznávania tváre a značky a ovládanie video pozadia. Obsahuje tiež okno pre výber kamery a vlákno pre rozpoznávanie tváre.
- **Aruco** - tento modul zabezpečuje funkcionalitu rozpoznávania značky z kamery pomocou knižnice Aruco vo vlastnom vlákne.

Moduly Aruco, OpenCV a QOpenCV sa prekladajú spoločne s programom iba v pripade, ak bola nájdená knižnica OpenCV. Modul Kinect potrebuje navyše knižnice OpenNI2 a NiTE2. Modul Speech vyžaduje prítomnosť Kinect SDK a Windows SDK.

Pôvodná aplikácia nie je na tieto nové moduly priamo naviazaná, a je ju možné skompilovať aj bez týchto knižníc, čo sa dosiahlo aj vhodnou úpravou súboru *CMakeLists.txt*.



Obr. 11: *Súčasná architektúra aplikácie*

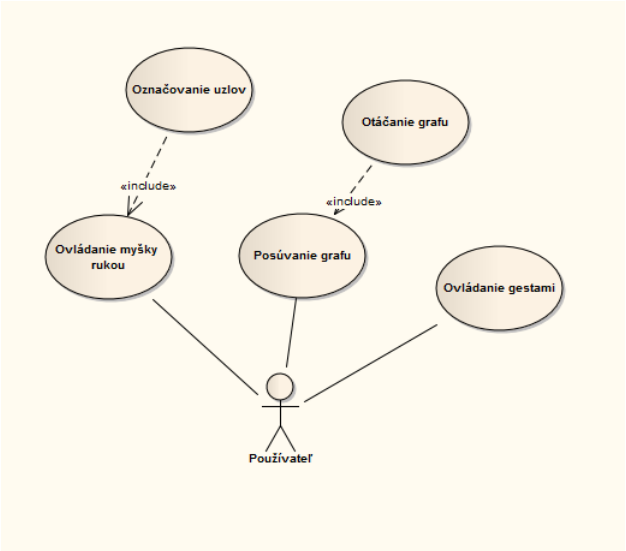
3.2.2 Novo pridané moduly

3.2.2.1 Modul rozpoznávania pohybu - Kinect

K jedným z najznámejších zariadení, ktoré umožňujú ovládanie v obohatenej realite patrí Kinect, preto v našom riešení zohráva dôležitú úlohu. Na implementáciu sme sa na rozdiel od klasického prístupu SDK Kinect knižníc od Windowsu rozhodli použiť open source riešenie OpenNI2 a na ňom založenú knižnicu NiTE2, ktorá je určená na prácu s rozpoznávaním objektov ako sú ruky, kostra človeka alebo vytvorenie hĺbkovej mapy priestoru. Táto knižnica bola vybraná z dôvodu, že je použiteľná na rôznych platformách operačných systémov (Linux, Windows, Mac). Túto knižnicu aktuálne (2014) odkúpil Apple, ale jej zdrojový kód je stále voľne dostupný.

Použitie

Kinect v programe je určený na viacero funkcií ako to je znázornené vo forme diagramu prípadu použitia na Obrázku 12. Medzi najvýznamnejšie patrí použitie rúk na ovládanie grafu a hýbanie myškou, taktiež významnú úlohu hrajú implementované gestá.



Obr. 12: *Diagram prípadov použitia*

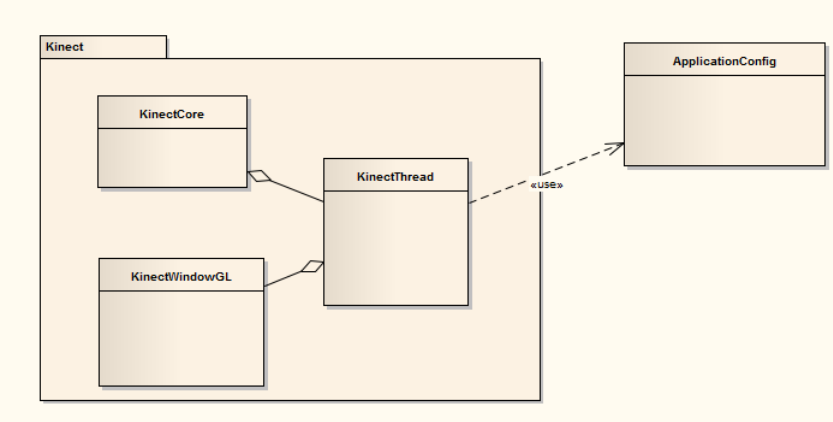
Návrhy použitia sú nasledujúce:

- *ovládanie myši* - na základe jednej ruky sme schopný ovládať myš, kde gestá ruky sú namapované na akcie myši,

- *ovládanie grafu* - na základe dvoch rúk vieme ovládať graf, pohybovať s ním a taktiež máme k dispozícií gestá na určité akcie s grafom.

Prvotná architektúra

Prvotná architektúra bola navrhnutá vzhľadom na možnosti knižnice NiTE ktorá dokázala zachytávať viacero rôznych typov objektov (mapa rúk, kostra ľudského tela, body z priestoru). Z tohto dôvodu bol prvý prototyp založený na architektúre, ktorá je znázornená na Obrázku 13, kde je základná trieda KinectCore- Singleton, ktorá vytvárala vlákno (KinectThread), v ktorom bežalo otvorenie spojenia s Kinect senzorom. KinectCore potom vytváral KinectWindowGL okno, ktoré využívalo grafickú knižnicu OpenGL, vďaka ktorému poskytovalo jednoduché vykresľovanie rôznych hĺbkových máp.

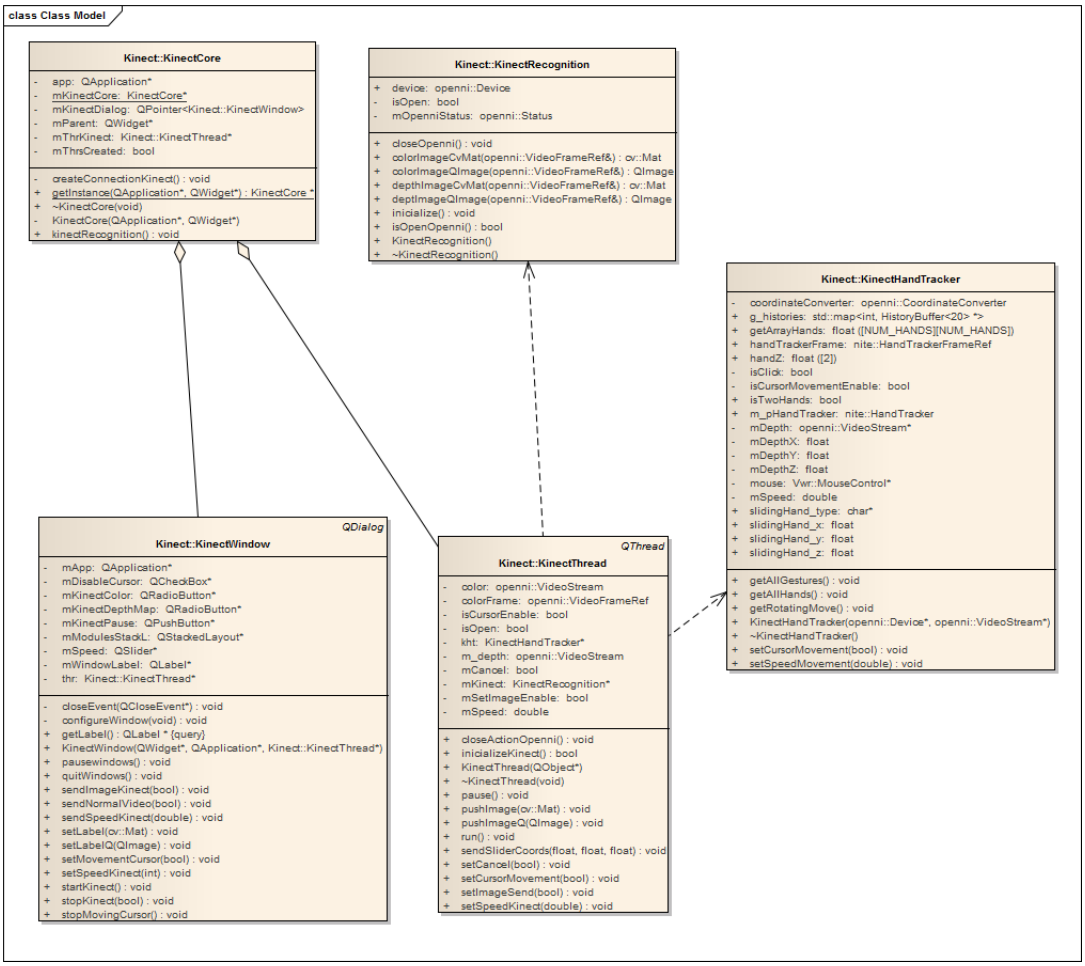


Obr. 13: *Pôvodne navrhnutá architektúra pre rozpoznávanie objektov s využitím Kinect knižníc*

Nevýhodou tohto riešenia bola komplikovaná komunikácia s OpenGL oknom a taktiež potreba behu až dvoch vlákien, jedno pre zber dát a počítanie (KinectThread) a taktiež OpenGL muselo bežať v samostatnom okne.

Finálna architektúra

Finálna architektúra vychádzala z pôvodne navrhnutej architektúry s pár zmenami. Pridalo sa tam QT okno (KinectWindow), ktoré pre svoj beh nepotrebuje samostatné vlákno. Následne po tom sa začali pridávať triedy na spracovanie dát, ktoré nám Kinect poskytoval. Táto architektúra je znázornená na Obrázku 14 ako diagram tried.



Obr. 14: *Súčasná architektúra Kinect modulu*

3.2.2.2 Zobrazovanie videa z Kinectu - KinectRecognition

Na rozdiel od OpenGL okien nám QT neposkytovalo vhodnú formu na zobrazenie hĺbkových máp do okien tohto formátu. Kvôli tomuto sme riešili zobrazenie nasledovne:

- *Klasický farebný obraz* - zo zariadenia sme získavali klasický dátový formát *openni::VideoStream*, ktoré sme následne museli transformovať do OpenCV formátu (*cv::Mat*), ktorý sme už v prípade FaceRecognition a Aruca používali na vykresľovanie do okna.

```
cv::Mat Kinect::KinectRecognition::
  colorImageCvMat(openni::VideoFrameRef &colorFrame)
{
  cv::Mat frame;
  const openni::RGB888Pixel* imageBuffer = (const openni::RGB888Pixel*)colorFrame.
      getData();
```

```
  frame.create(colorFrame.getHeight(), colorFrame.getWidth(), CV_8UC3);
  memcpy(frame.data, imageBuffer, 3*colorFrame.getHeight()*colorFrame.getWidth()*
      sizeof(uint8_t));

  return frame;
}
```

- *Hĺbkový obraz* - Ako v predchádzajúcom prípade, použili sme OpenCV formát s tým, že sme si ho prispôsobili vykresľovaniu kvôli znázorneniu blízkosti a vzdialeností objektov.

Pri optimalizácií tohto riešenia sme využili klasický formát QT - *QImage*, ktorý je oveľa vhodnejší na posielanie a zobrazovanie v rámci aplikácií bežiacich na QT architektúre.

```
QImage Kinect::KinectRecognition::
  colorImageQImage(openni::VideoFrameRef &colorFrame)
{
  QImage image = QImage((uchar*)colorFrame.getData(), colorFrame.getWidth(), colorFrame.
      getHeight(), QImage::Format_RGB888);
  image = image.rgbSwapped();
  return image;
}
```

Manipulácia s grafom a základné gestá - KinectHandTracker

Na samotnú interakciu za použitia rúk sme využili dostupné možnosti knižníc OpenNI a NiTE. V závislosti od vyťaženia systému, sme zo senzora získavali 15-25 obrazov za sekundu. Pre detekciu v priestore je potrebná hĺbková mapa, v ktorej cez funkciu (*getGestures*) získavame polohu snímaných rúk po vykonaní gesta (kliknutie – pohyb vpred a vzad po osi y, kývnutie – kmitavý pohyb doľava a doprava po osi x).

Detekcia prebieha v samostatnom vlákne (*KinectThread*), triedou *getAllHands()* indexujeme všetky ruky v snímanom priestore a priraďujeme im identifikátor pre ďalšiu prácu s nimi. Keďže každá priestorová snímka obsahuje odlišnú pozíciu rúk (v závislosti na aktivite a frekvencii snímok, rozdiel milimetre až centimetre) bolo nutné implementovať mechanizmus na určenie, resp. párovanie rúk s predchádzajúcou snímkou. Riešením je trieda *HistoryBuffer*, ktorú sme využívali s hĺbkou histórie 20 snímok. Vďaka implicitnému monitorovaniu polohy získavame aj zmenu polohy, teda pohyb rúk do jednotlivých smerov. Túto funkcionalitu rieši funkcia *getRotatingMove*, ktorá premieňa markantnejšie zmeny polohy do boolovských údajov o pohybe.

Poloha rúk priamo vplýva na proces ovládania v reálnom čase, rovnako teda aj na zobrazenie v obohatenej realite pred používateľom. Pomocou polohy dvoch rúk vytvárame pomyselný obdĺžnik, ktorý vo funkcii "selectöznačuje skupiny uzlov. Funkcionalitu spojenú

s detekciou zároveň môžeme overiť v samostatnom okne, ktoré doplňujeme pomocou OpenCV o základné tvary a texty. Takto môžeme vidieť rám "selectu"tvorený OpenCV triedou *Rect*, alebo znázornený smer pohybu pomocou textu.

Pre získanie spracovateľných dát polohy rúk je potrebná konverzia získaných informácií zo senzoru Kinect na dáta reprezentujúce reálne prostredie. Funkcia *convertWorldToDepth* poskytuje prepočet pozície ruky z milimetrových údajov na údaje reprezentované pixelmi, touto konverziou vieme presne určiť pozíciu detegovanej ruky a tieto výstupy slúžia následne ako vstup pre samotnú manipuláciu s grafom.

```
this->m_pHandTracker.convertHandCoordinatesToDepth(position1.x, position1.y, position1.z
    , &this->slidingHand_x, &this->slidingHand_y, &this->slidingHand_z);

coordinateConverter.convertWorldToDepth(*mDepth, user.getPosition().x,, user.getPosition
    ().y, user.getPosition().z, &this->slidingHand_x, &this->slidingHand_y, &this->
    slidingHand_z);
```

Ovládanie myši na základe Kinectu - MouseController

Prvé riešenia na ovládanie myši boli založené na Windows API knižniciach, ktoré boli dostupné v systémoch na platforme Windows.

```
#include <winuser.h>

mouse_event(MOUSEEVENTF_MOVE | MOUSEEVENTF_ABSOLUTE, x, y, 0, 0);
mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
```

Tie najlepšie výsledky nám ponúkala práve WinApi na zachytávanie akcií kurzora a ich jeho následnú manipuláciu.

```
POINT * ptr = GetCursorPos() // get position mouse
POINT points = *ptr;
xCoord = points.x; // x cursor coordinates
yCoord = points.y; //y cursor coordinates
```

Žiaľ, žiadne z týchto riešení nebolo multiplatformové, ale závislé na platforme Windows. Finálnym riešením na stalo obmedzené použitie myši založené na *osgViever:Viewer*, ktorý ponúkal ovládanie obmedzené ovládanie len na widgete, ktorý sa staral o vykresľovanie. Toto nám dovoľovalo v tom konkrétnom okne:

- *manipuláciu s kurzorom,*

```
cursor().pos();
cursor().setPos();
```

- *zistenie o okne,*

```
window()->pos();
window()->grabMouse();
```

- *manipuláciu s akciami.*

```
this->getEventQueue()->mouseButtonPress;
this->getEventQueue()->mouseButtonRelease;
```

A mnoho ďalších akcií, ktoré nie sú tak podstatné ako práve tieto. Na základe tohto sme navrhli funkcie, ktoré sme potom používali. Základný princíp spočítal v krokoch:

1. poskytnutie pozície z Kinectu,
2. prepočet pozície z Kinectu na rozmery, na ktorej to bolo premietané,
3. prepočet skutočnej pozície v rámci okna programu,
4. posunutie kurzora,
5. v prípade akcií na okne bol potrebný prepočet do *osgViewer* pozícií.

Na základe tohto sme navrhli viacero funkcií, medzi najznámejšie patria posun kurzora a akcie spojené s tlačidlami myšky.

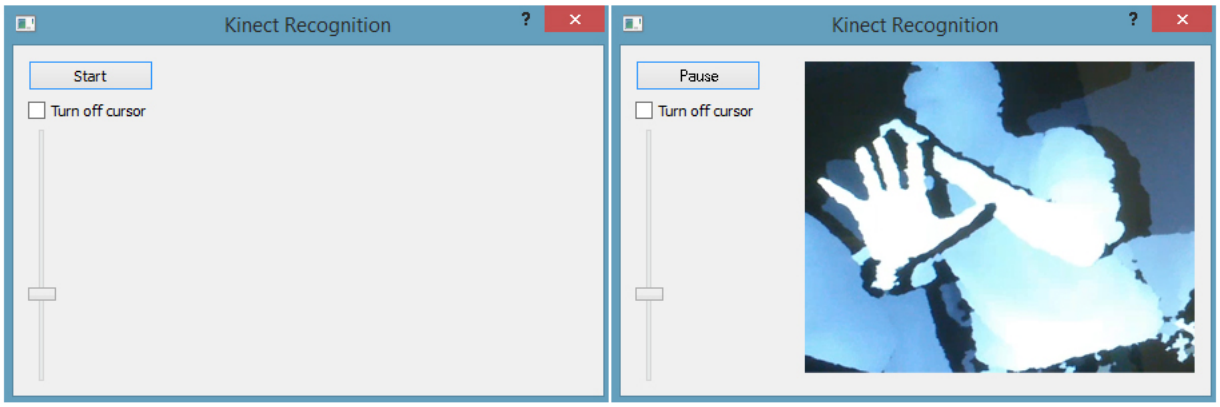
```
/**
   * @brief move cursor of mouse at concrete place
   * of screen based on old position of mouse
   * @param positionX new position x of cursor
   * @param positionY new positino y of cursor
   * @param isClick is press any button
   */
  void moveCursorMouse(double positionX,
     double positionY, bool isClick);

/**
   * @brief click and press of button at
   *position os cursor
   * @param positionX position X of click
   * @param positionY position Y of click
   * @param button type of mouse button
   */
  void clickPressMouse(float positionX, float positionY, Qt::MouseButton button);
```

Na získanie údajov na pohyb sme používali predchádzajúce funkcie detekcie rúk, do ktorej sme si definovali získanie jednej ruky a následne na základe jej pohybu sme získavali pozíciu pre pohyb a taktiež akcie pre tlačidlá myší.

Výsledné okno

Okno (zobrazené na Obrázku 15) pre informácie o vykonávaní a detekcii vo vlákne KinectThread sa nutne zobrazuje pri zapnutí a vypnutí senzora a tiež podľa potreby používateľa pri interakcii s programom.



Obr. 15: *Výsledné okno KinectThreadu*

3.2.2.3 Modul rozpoznávania hlasu - Kinect Speech

Ďalším prvkom obohatenej reality v systéme zabezpečujeme pomocou zvukových príkazov. Požiadavka pre danú funkcionalitu bola hlavne v presnosti rozpoznávania hlasových povelov z dostatočnej vzdialenosti, nakoľko sa nachádzame v prostredí do 2 metrov od fólie, na ktorú vykresľujeme.

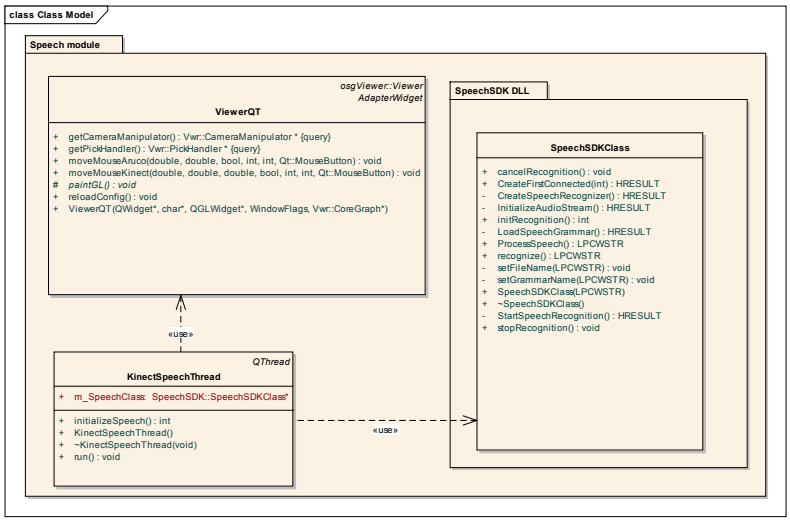
Pre vytvorenie funkcionality sme použili zariadenie Kinect a jeho knižnicu Kinect SDK pod operačným systémom Windows. Moduly pre zvuk a dalšie funkcionality sa nachádzali v samostatnej knižnici Speech SDK, ktorú bolo potrebné dodatočne nainštalovať.

Okrem základnej požiadavky pre rozpoznanie hlasových povelov bolo potrebné zabezpečiť aj nasledovné podmienky:

- pridávanie nových povelov musí byť jendoduché,
- zmena kódu pri nových poveloch čo najmenšia,
- celkový modul rozpoznávania čo najjednoduchší.

Na základe predchádzajúcich požiadaviek sa vytvoril samostatný modul pre rozpoznávanie dostupný v dynamickej (DLL) knižnici. V hlavnom programe bolo vytvorené samostatné vlákno pre potreby inicializácie a vykonávania tak, aby s touto dynamickou knižnicou bolo schopné komunikovať cez triedu *SpeechSDKClass*.

Hlavným dôvodom vytvorenia danej knižnice boli aj rozsiahlejšie úpravy potrebné pre implementáciu, ktoré by pre účel aplikácie nebolo potrebné meniť a spôsobili by väčšiu neprehľadnosť kódu. Pridávanie nových povelov je zabezpečené cez XML dokument a potreba zásahu do kódu je minimálna.



Obr. 16: *Diagram tried modulu Speech*

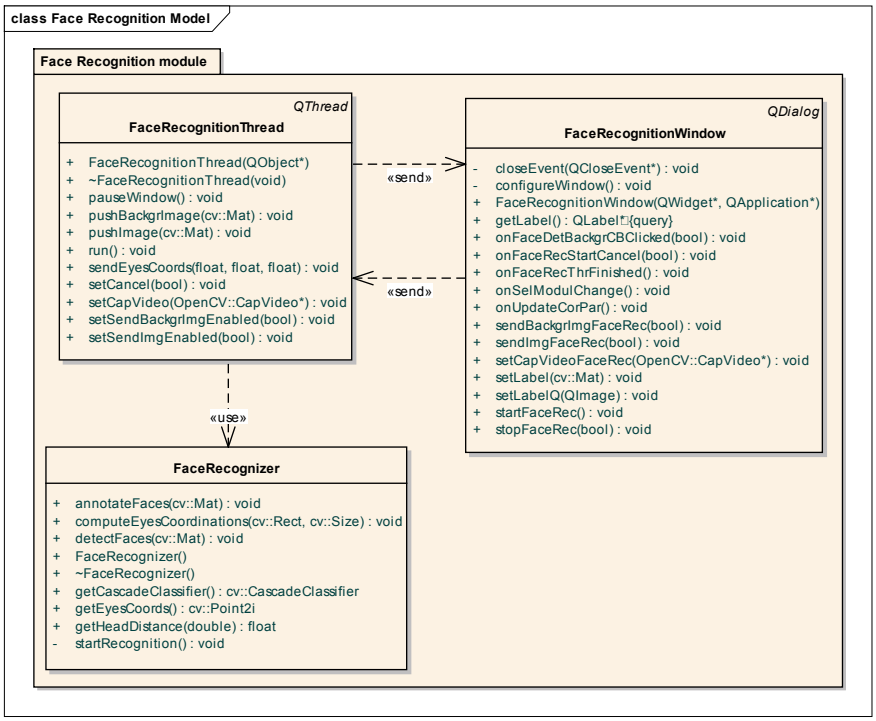
Rozpoznávanie povelov je teda možné vykonať jednoduchým porovnaním stringu. Krátke znázornenie rozpoznávania vyjadruje nasledovná ukážka z kódu:

```
const wchar_t *a= this->m_SpeechClass->recognize();
  if (0== wcscmp(a,L"ALL")){
    qDebug() << "command: select all nodes";
    AppCore::Core::getInstance()->getCoreWindow()-> multiSelectClicked(true);
    QOSG::ViewerQT *viewer=AppCore::Core::getInstance()->
    getCoreWindow()->GetViewerQt();
    if(viewer!=NULL) {
      int mWindowWidth=viewer->width();
      int mWindowHeight=viewer->height();
      viewer->getEventQueue()->mouseButtonPress(0,0, Qt::LeftButton);
      viewer->getEventQueue()->
      mouseMotion(mWindowWidth,mWindowHeight);
      viewer->getEventQueue()->
      mouseButtonRelease(mWindowWidth,mWindowHeight, Qt::LeftButton);
    }
  }
  if (0== wcscmp(a,L"SPHERE")){
    qDebug() << "command: sphere";
    AppCore::Core::getInstance()->getCoreWindow()->
    setRestriction_Sphere();
  }
```

3.2.2.4 Modul rozpoznávania tváre - FaceRecognition

Modul pre rozpoznanie tváre používateľa. Na základe rozpoznanej tváre sa vykonáva transformácia vykresleného grafu. Základné časti tohto modulu sa skladajú z nasledujúcich tried (Obrázok 17 je príslušný diagram tried tohto modulu):

- *FaceRecognizer* - nachádza sa tu implementácia rozpoznania tváre ako aj ďalšie funkcie ako je anotácia tváre, prípadne výpočet polohy očí.
- *FaceRecognitionThread* - samostatné vlákno sa spustí po spustení rozpoznávania tváre. Riadi činnosť rozpoznávania počas jeho behu.
- *FaceRecognitionWinow* - trieda určená pre zobrazovanie grafického výstupu modulu rozpoznávania. Spolupracuje s vláknom. Príjima výstupné obrázky a obsahuje funkcionalitu v podobe Signal/Slot komunikácie cez Qt rozhranie.



Obr. 17: *Diagram tried modulu FaceRecognition*

Vytvorenie vlákna riadi trieda *OpenCVCore*, v ktorej sa nachádzajú aj iné funkcionality podobného typu.

Samotná funkcionalita rozpoznávania je implementovaná s pomocou OpenCV knižnice cez klasiikátor na základe *haar like* príznakov. Klasifikátor sa najprv potrebuje tvár naučit

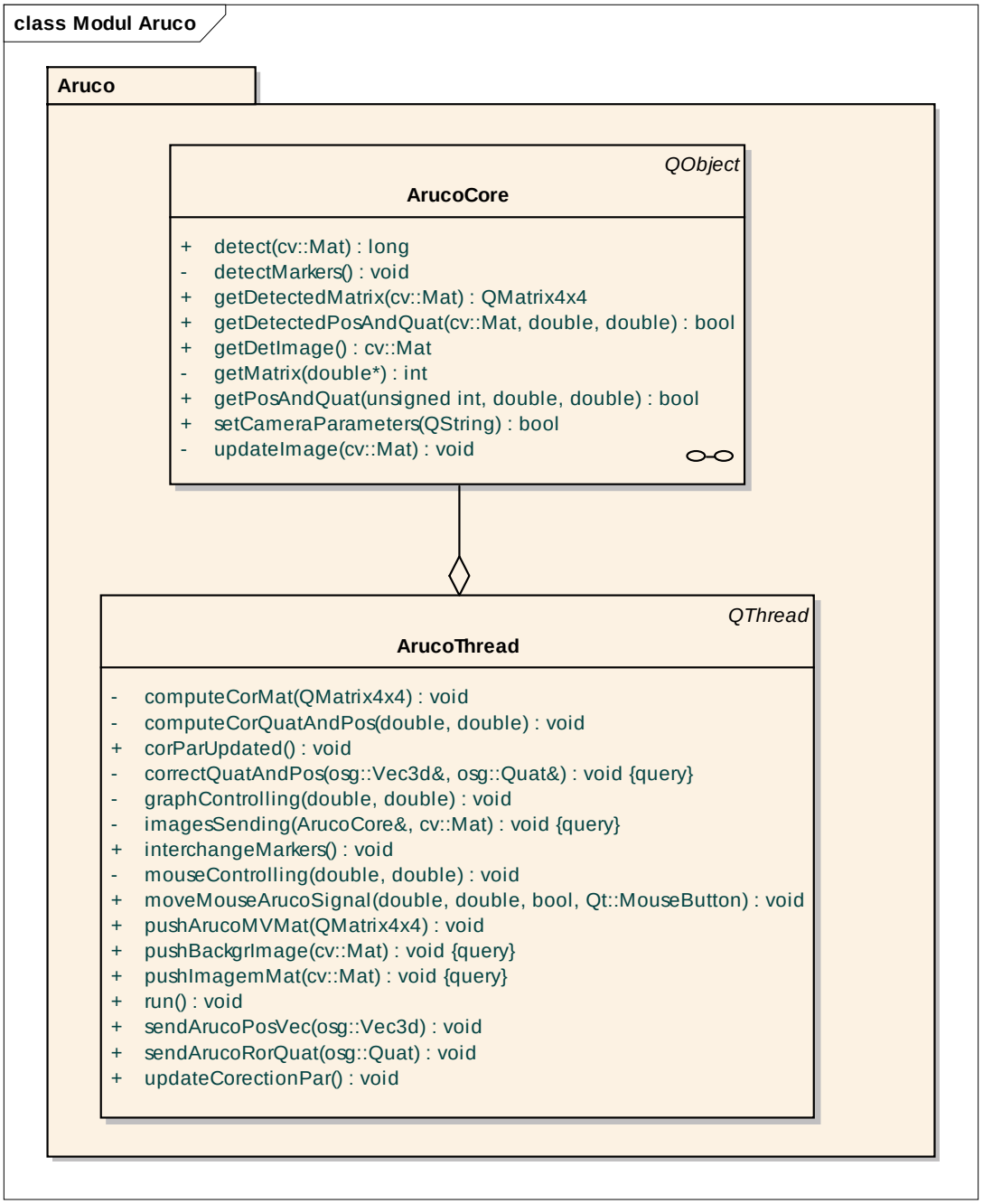
pred samotným rozpoznaním. OpenCV knižnica disponuje samostatným XML súborom s naučenými príznakmi, ktoré sa pre danú funkcionalitu využívajú.

Klasifikátor je kaskádový, čo znamená, že celkový klasifikátor použitý pre detekciu tváre je zložený z viac menších rýchlejších klasifikátorov. Tým zabezpečíme vykonávanie v reálnom čase, ktoré ale stále má svoje nedostatky v podobe rotačnej invariancie, ktorú implementácia neposkytuje.

Implementácia vyžadovala pomerne vysoký výkon procesora a bola preto ďalej upravovaná. Boli vykonané nasledovné kroky:

- po prvej detekcii tváre nasledovné detekcie prebiehali už len v časti obrázku, kde bola tvár detegovaná predtým,
- ako pohyb sa vyhodnotil až keď sa novo detegovaná tvár vzdialila od pôvodnej o 10%.

3.2.2.5 Modul rozpoznávania značky - Aruco



Obr. 18: *Diagram tried modulu Aruco*

Tento modul obsahuje triedy znázornené na Obrázku 18 zabezpečujúce rozpoznávanie značiek a spracovanie získaných informácií o ich polohe a orientácií. Používateľ má tak k

dispozícii jednoduchý spôsob interakcie s grafom, pričom môže intuitívnejšie otáčať a obzerať si graf pomocou značky, než klasicky počítačovou myšou. Tiež sme implementovali jednoduché metódy pre ovládanie kurzora značkou.

Obsahuje triedy:

- *ArucoCore*,
- *ArucoThread*.

Trieda *ArucoCore* poskytuje funkcionalitu pre rozpoznanie značky, pričom narába s funkciami knižnice Aruco. Je vytvorená ako objekt, ktorému musíme na začiaktu nastaviť parametre kamery. Následne využívame hlavne metódy *detect()* a *getPosAndQuat()*. Prvej najskôr predáme obraz, v ktorom chceme detegovať značky a druhou následne získame informáciu o polohe a orientácii zvolenej značky.

Tiež poskytuje možnosť dokresliť kocku na obraz, z ktorého ju detegovala a predať ho funkciou, čo neskôr využívame pre kontrolu detekcie.

```
long ArucoCore::detect(cv::Mat inputImage)
{
  mCamParam.resize(inputImage.size());
  mCamImage = inputImage;

  detectMarkers();
  return mMarkers.size();
}

bool ArucoCore::getPosAndQuat(unsigned int markerNum, double position
    [3],
  double quaternion[4])
{
  // markerNum is counted from 0
  if( mMarkers.size() > markerNum){
    mMarkers[markerNum].OgreGetPoseParameters( position, quaternion);
    return true;
  } else {
    return false;
  }
}
```

ArucoThread

Triedu *ArucoThread* sme navrhli tak, aby tvorila samostatné vlákno pre funkcionalitu rozpoznávania značky. Na detekciu používa práve objekt triedy *ArucoCore*. V metóde *run()* sa vykonáva jej hlavná funkcionalita. Tou je skontrolovať či je nastavená kamera, pripraviť parametre pre korekciu výslednej informácie o polohe a orientácii. Následne v cykle, až do prerušenia opakuje nasledujúce kroky:

1. získanie obrazu z kamery a jeho poslanie na detekciu do objektu ArucoCore.
2. Získanie pozície a rotácie prvej značky. Ich korekcia a následné odoslanie pre posunutie grafu a otočenie kamerou alebo grafom.
3. Získanie pozície a rotácie druhej značky a ich úprava. Navrhli sme, že dve rôzne orientácie značky môžu predstavovať klik myši a jej poloha polohu kurzora. Takto získané informácie ďalej posielame ďalej pre ovládanie kurzora myši.
4. Ak je zapnuté video pozadie, alebo video v ovládacom okne *FacerecognitionWindow*, obraz s doplnenou kockou sa posiela ďalej na jeho zobrazenie.

Chýbajúce funkcie:

- Aruco umožňuje detegovať množstvo značiek súčasne, naša implementácia využíva zatiaľ len dvojicu.
- Pre simulovanie kliknutia využívame rotáciu, avšak je možné doplniť ešte rozlišovanie medzi jednotlivými typmi tlačidiel myši.

3.2.2.6 Spoločné ovládania modulov FaceRecognition a Aruco

Kvôli podobnosti modulu pre rozpoznávanie značky s modulom pre detekciu tváre, sme pre ne navrhli niekoľko spoločných tried zobrazených na Obrázku 19. Sú to: spoločné používateľské rozhranie, spoločný kontrolér, rozhranie pre voľbu kamery a spoločný manažér kamier. Tieto triedy sú v moduloch *OpenCV* a *QOpenCV*. Konkrétne sú to:

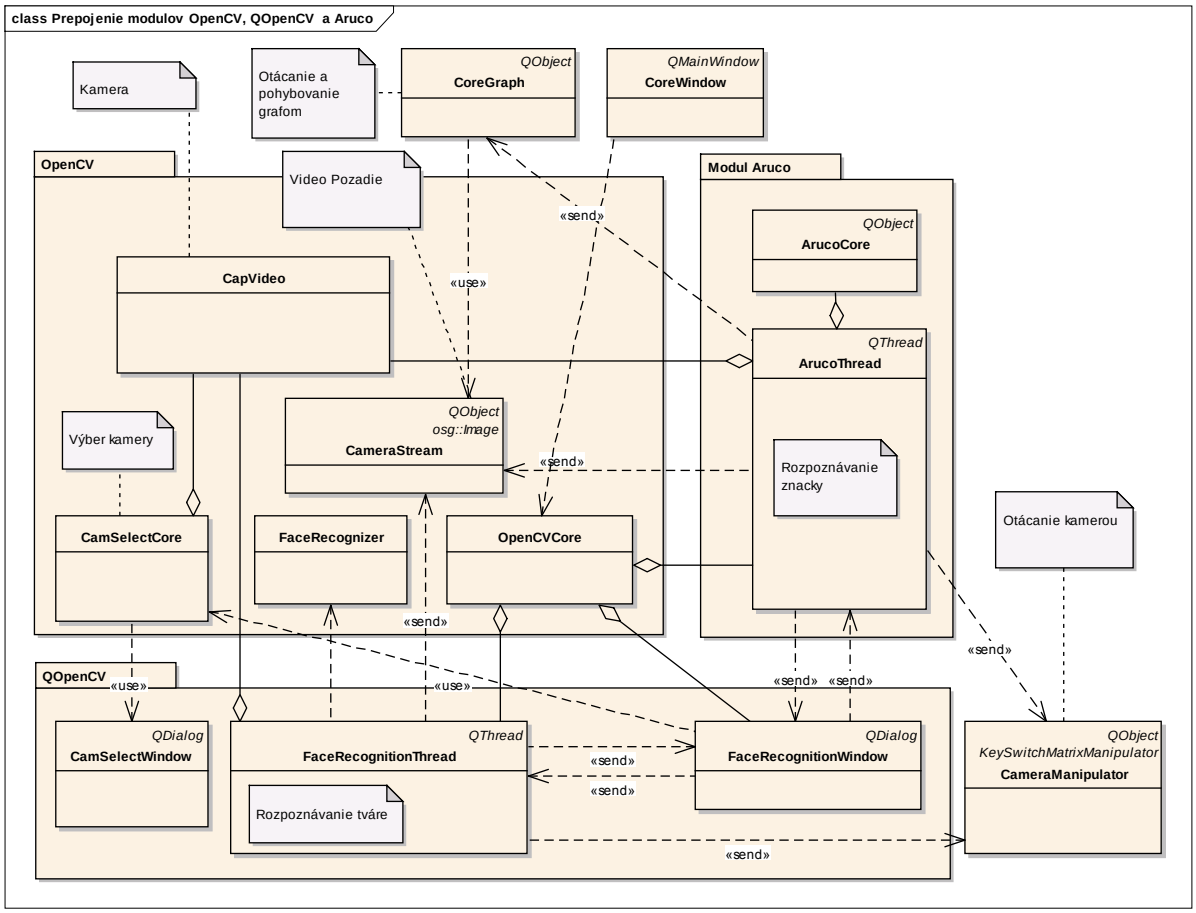
- **OpenCVCore** - spoločný kontrolér pre rozpoznávanie značky a tváre navrhnutý podľa vzoru Singleton. Vytvára inštancie oboch vláken a okna používateľského rozhrania. Zabezpečuje všetky prepojenia vlákien s rozhraním, aby neboli navzájom závislé. Tiež vytvára prepojenia medzi vláknami a triedami *CameraManipulátor, CoreGraph*, aby bolo možné odosielať informácie pre otáčenie a pohybovanie grafom a kamerou, a tiež aktualizáciu video pozadia.
- **FaceRecognitionWindow** - spoločné používateľské rozhranie pre ovládanie rozpoznávania značky a tváre vo forme dialógu. Používateľ má možnosť ovládať

nezávisle obe funkcionality. Vie spúšťať osobitne vlákna a nastavovať pre ne parametre. Má možnosť zobraziť si video pre každé vlákno alebo zavrieť okno, aby ho neobmedzovalo, pričom však vlákna bežia naďalej.

- **CamSelectCore** - keďže používateľ môže mať k dispozícii len jednu kameru, pričom však rozpoznávanie značiek a tváre sú bežiace v nezávislých vláknach, navrhli sme *Singleton* triedu, ktorá sa stará o manažovanie a správu kamier. Táto trieda zabezpečuje zistenie počtu dostupných kamier a poskytuje metódy pre ich získanie, zdieľanie a uvoľnenie. Preto si uchováva ich zoznam aj s ich aktuálnymi parametrami. Takto je možné nastaviť kameru pre rozpoznávanie značky a v prípade potreby, použiť rovnakú kameru súčasne aj pre rozpoznávanie tváre so zachovaním rovnakého rozlíšenia. Kamery sú reprezentované formou objektov triedy *CapVideo*.
- **CamSelectWindow** - je to modálny dialóg používateľského rozhrania pre výber a nastavenie kamery používaný *CamSelectCore* triedou. Implementuje jednoduchú tabuľku, kde je možné zvoliť kameru a nastaviť pre ňu rozlíšenie.
- **CapVideo** - predstavuje objekt kamery a poskytuje ďalšie doplňujúce funkcie pre jej ovládanie, napr. aby mohli kameru využívať obe vlákna súčasne.

```
void OpenCV::CapVideo::startCamera(int width,int height){
  this->connections++;
  if ( !capture.isOpened() ){
    cv::VideoCapture cap( this->device_id);
    this->capture = cap;
    this->capture.set(CV_CAP_PROP_FRAME_WIDTH, width);
    this->capture.set(CV_CAP_PROP_FRAME_HEIGHT, height);
    this->width = (int) this->capture.get(CV_CAP_PROP_FRAME_WIDTH
        );
    this->height = (int) this->capture.get(
        CV_CAP_PROP_FRAME_HEIGHT);
    return;
  }
}

void OpenCV::CapVideo::release(){
  this->connections--;
  if (this->connections == 0){
    this->capture.release();
    this->width = 0;
    this->height = 0;
  }
}
```



Obr. 19: *Diagram tried znázorňujúci prepojenie modulov*

3.2.3 Zmeny v pôvodných moduloch a nová funkcionalita

V tejto časti opisujeme hlavné vykonané zmeny v pôvodných moduloch aplikácie. Niektoré zmeny opravovali chyby, ale väčšina z nich súvisela priamo s funkcionalitou novo pridaných modulov.

3.2.3.1 Úprava stromu scény OpenSceneGraphu

Hlavným zmyslom OpenScenGraphu je reprezentácia scény formou stromovej štruktúry. V pôvodnej verzii aplikácie bol tento princíp značne nedodržaný. Scénu tvoril koreňový uzol, ktorý mal priamo ako svojich potomkov zoznamy hrán, multihrán, uzlov, metauzlov atď. a medzi nimi aj pozadie. Toto však neumožňovalo jednak meniť pozadie, nakoľko to bolo viazané na poradie a tiež ani manipulovať s grafom. Preto sme navrhli zmenu vo forme oddelenia časti grafu od pozadia ako potomkov samostatného uzla reprezentujúceho graf. Toto umožnilo jednak pridanie rôznych druhov pozadia, ale hlavne aj otáčanie a posúvanie

grafom formou transformačného uzla, čo predtým implementované nebolo.

```
root = new osg::Group();
graphRotTransf = new osg::MatrixTransform();
graphGroup = new osg::Group();

graphRotTransf->addChild(graphGroup);
root->addChild(graphRotTransf);

// backgroung this must be last Node in root !!!  ( because of ortho2d
    background)
root->addChild( createBackground());
```

3.2.3.2 Pridanie možnosti rotovať a posúvať graf

Oddelením grafu do samostatného uzla bolo umožnené pridanie otáčania a pohybovania s grafom. To bolo potrebné, nakoľko pri video pozadí je dôležité zachovať kameru v pôvodnom stave a neotáčať s ňou. U druhého to potrebné nie je. Je to implemetované formou transformačného uzla pred uzlom grafu, ktorý je možné meniť pomocou nasledujúcich slotov, ktoré sú využité práve aj pre otáčanie, buď pomocou značky, tváre, myšou alebo všetkými aj súčasne.

```
void CoreGraph::updateGraphRotByAruco(const osg::Quat quat ) {
  mRotAruco = quat;
  computeGraphRotTransf();
}

void CoreGraph::updateGraphRotByMouse( const osg::Quat quat ) {
  mRotMouse = mRotMouse * quat;
  computeGraphRotTransf();
}

void CoreGraph::updateGraphRotByFaceDet( const osg::Quat quat ) {
  mRotFaceDet = quat;
  computeGraphRotTransf();
}

void CoreGraph::computeGraphRotTransf() {
  osg::Matrixd graphTransfMat( mRotMouse * mRotAruco * mRotFaceDet );
  graphRotTransf->setMatrix(graphTransfMat);
}
```

3.2.3.3 Otáčania kamery pomocou pohybov, pozície hlavy alebo značkou

Pomocou tváre a gest je možné otáčať okrem grafu aj kamerou, čo bolo prvotným zámerom a je to aj prednastavené správanie. Aby bolo možné využívať oba spôsoby súčasne, pre každý z nich je implementovaný osobitný slot v triede *CameraManipulator*, kde posielajú pozíciu a vzdialenosť, z čoho sa vypočíta otočenie cez quaterniony. Najskôr sa vypočítava aktuálne otočenie, ktoré sa pravidelne aktualizuje.

```
// both rotation must be separated
// horizontal rotation
trackball(axis,angle, x, 0.0, 0.0, 0.0);
osg::Quat Xnew_rotate(angle * throwScale,axis);

// vertical rotation
trackball(axis,angle,0.0, y, 0.0, 0.0);
osg::Quat Ynew_rotate(angle * throwScale,axis);

_rotHeadFaceDet = Xnew_rotate * Ynew_rotate;
```

Kvôli možnosti otočiť kameru ešte viac bola implementovaná schopnosť otáčať ju súvisle, ak sa dostaneme gestom alebo hlavou blízko ku niektorému z okrajov. Toto otočenie postupne kumuluje a dovoľuje otočiť kameru úplne akokoľvek.

```
float region = this->appConf->getValue("Kinect.RegionRotationTreshold")
    .toFloat();
float step = this->appConf->getValue("Kinect.RegionRotationStep").
    toFloat();
if( x < -region || x > region ){
  aux = x < 0.0f ? -step : step;
  trackball(axisAux, angleAux, aux, 0.0, 0.0, 0.0);
  osg::Quat rotHorAux = osg::Quat(angleAux * throwScale, axisAux);
  _rotationAux = _rotationAux * rotHorAux;
}
...
_rotationHead = _rotationAux * _rotHeadFaceDet * _rotHeadKinect;
```

Každé otáčanie je ale závislé na aktuálnom otočení kamery, aby sa vykonalo v správnom smere. Preto aj pri otáčaní grafom namiesto kamery, sa toto otočenie vypočíta najskôr pomocou týchto slotov, a až následne, buď ovplyvní kameru, alebo pomocou signálov zavolajú vyššie opísané sloty v *CoreGraph* triede.

Nedokončená funkcionalita:

Pre lepší efekt rozšírenej reality a pocitu, že graf sa nachádza v priestore za projekčným plátnom, bola implementovaná metóda *updateProjectionAccordingFace()*. Jej úlohou bolo upravovať perspektívnu projekciu, čo je podrobnejšie vysvetlené v časti dokumentácie zo zimného semestra. Je prispôsobená skôr pre používateľa pred displejom notebooku a väčšej vzdialenosti ako pre použitie projekčného plátna - v tomto prípade nefunguje správne a je ju potrebné upraviť.

3.2.3.4 Video pozadie z kamery

Okrem 2 pôvodných pozadí Skybox a Noise pozadia, boli pridané možnosť zobraziť čierne pozadie, čiže žiadne, čo je dôležité pri použití projekčného plátna. Ďalej aj funkcia video pozadia, čo môže pomôcť simulovať priehľadné projekčné plátno, napr. umiestnením kamery za monitor. Implementované bolo 2 spôsobmi:

- prvý spôsob ho dosahuje obdĺžnikom v 3D scéne potiahnutým textúrou. Po úprave má tento spôsob výhodu v zachovaní pomeru strán videa, avšak pri roztiahnutí vznikajú na bokoch čierne pásy, kde sa nezobrazuje video. Tiež neumožňuje otáčať kamerou. Je implementovaný v metóde *createTextureBackground()*.
- Druhý, ktorý odstraňuje tieto nedostatky je implementovaný v metóde *createOrtho2dBackground()* triedy CoreGraph. Je vytvorený v Ortho 2D režime, čo spôsobuje, že sa rozťahuje a sťahuje podľa šírky obrazu, čo môže spôsobiť jeho deformáciu. Avšak, toto je pri normálnej veľkosti okna zanedbateľné.

Jednotlivé typy pozadia je možné prepínať pomocou konfiguračného súboru.

Dôležitým článkom video pozadia je objekt triedy **CameraStrean** modulu OpenCV, ktorý je v oboch prípadoch využívaný ako obrázok pre vytvorenie textúry. Tento je špeciálny v tom, že je odvodený zároveň od osg::Image a QObject, vďaka čomu podporuje sloty, čo umožňuje ho neustále meniť a zároveň sa po každej zmene aktualizujú všetky textúry, ktoré ho využívajú. Náročná bola konverzia z cv::Mat do osg::imageData, ktorá funguje rôzne na Windows a Linux platformách.

```
// There will be probably needed refactoring on MAC OS
#ifdef WIN32
cvCopy(&(IplImage)cvImg, iplImg, NULL);
setImage( iplImg->width, iplImg->height,
       3, GL_RGB, GL_RGB,
       GL_UNSIGNED_BYTE,
       (unsigned char*) iplImg->imageData,
```

```
        osg::Image::NO_DELETE, 1);

cvImg.~Mat();

#else
cv::Mat img = cvImg.clone();
setImage(img.cols, img.rows,
        8, img.channels(), GL_RGB,
        GL_UNSIGNED_BYTE,
        (unsigned char*)(img.data),
        NO_DELETE, 1);
#endif
```

3.2.3.5 Fullscreen režim

Jednoduché, no užitočné pri použití projekčného plátna bolo pridanie fullscreen módu. Úprava bola vykonaná v triede *PickHandler*.

```
if( AppCore::Core::getInstance()->getCoreWindow()->isFullScreen()){
  AppCore::Core::getInstance()->getCoreWindow()->menuBar()->show();
  AppCore::Core::getInstance()->getCoreWindow()->statusBar()->show();
  AppCore::Core::getInstance()->getCoreWindow()->showNormal();

  if(hideToolbars){
    QList<QToolBar *> toolbars = AppCore::Core::getInstance()->
    getCoreWindow()->findChildren<QToolBar *>();
    QListIterator<QToolBar *> i(toolbars);
    while (i.hasNext()){
    i.next()->show();
    }
  }
} ....
```

3.2.3.6 Zobrazenie rôznych textúr pre uzly

Táto zmena bola vykonaná hlavne pre zatraktívnenie nášho projektu. Jej cieľom bolo ukázať, že náš projekt má aj veľké praktické využitie. Bolo potrebné vymeniť základnú textúru, ktorá sa používa pre bežný uzol. Úprava bola vykonaná iba pre graf s názvom *Veolia*, ktorý je uložený v súbore "Veolia.graphml". Tento graf zobrazuje súbor francúzskych domén Veolia. Jednotlivé domény sú reprezentované zakladanými uzlami a ich hrany zobrazujú vzťahy medzi nimi.

V prvom rade boli stiahnuté obrázky domén, na ktorých sa následne upravilo rozlíšenie, formát a názvy. Názvy sú upravené tak, aby ich súčasťou bolo identifikačné číslo, ktoré je unikátne pre každý uzol.

Zmena v zdrojovom kóde sa týkala hlavne triedy *Node*, ktorá sa nachádza v module *Data*. Táto trieda sa volá vždy, keď sa vytvára alebo upravuje akýkoľvek uzol grafu. V prvom kroku musíme overiť, či sa jedná o bežný uzol, potom podľa identifikačného čísla vytvoríme reťazec, v ktorom je uložená cesta k obrázku. Ten je následne použitý ako textúra pre nový uzol.

```
if(name.compare("PNode")!=0)
{
  QString path = "\\img2\\";
  path.append(QString::number(id-3));
  path.append("_veolia.png");

  qDebug()<< path;

  osg::ref_ptr<osg::Texture2D> texture =
  Vwr::DataHelper::readTextureFromFile(path);
  stateSet->setTextureAttributeAndModes(0,texture, osg::StateAttribute
      ::ON);

}
else
{
  stateSet->setTextureAttributeAndModes(0,type->getTypeTexture(),
  osg::StateAttribute::ON);
}
```

3.3 Nedokončená funkcionalita

3.3.1 Rozpoznanie scény

V rámci záverečných šprintov sme sa pokúsili implementovať funkcionalitu rozpoznania scény za grafom, čo sa nám, žiaľ, nepodarilo dokončiť do termínu odovzdania.

Hlavná myšlienka

Rozpoznanie scény za grafom nám umožňuje pracovať s objektami v scéne tak ako by sme vytvorili pocit reálneho grafu v reálnej scéne. Ak poznáme jednotlivé objekty v scéne, graf s nimi môže jednoducho interagovať napríklad tak, že používateľ môže tento graf položiť na stôl alebo rozprestrieť na stenu. Zároveň by tieto objekty fungovali ako obmedzenia pre graf, takže by sa graf samotný zastavil napr. pred stenou miestnosti a ďalej by sa nepohyboval.

Hĺbková mapa

Pre vytvorenie scény sme sa rozhodli využiť hĺbkovú mapu scény vytvorenú pomocou senzoru Kinect. Kinect nastavíme tak, aby snímal scénu a takto získame hĺbkovú mapu, ktorú ďalej spracovávame. Keďže sa scéna zriedka mení, stačí túto snímku vytvoriť iba raz - pri spustení programu, rovnako ako aj vytvoriť mapu samotnú. Mapa je uložená v matici, kde sa na jednotlivé pozície ukladá hodnota z v milimetroch.

RANSAC algoritmus

Z hĺbkovej mapy následne získame roviny jednotlivých objektov. Tieto roviny sú získané pomocou algoritmu, ktorý sme implementovali. Tento algoritmus je založený na RANSAC algoritme.

V rámci výpočtu roviny vyberieme z hĺbkovej mapy tri náhodné body. Uvažujeme pritom, že hodnoty x a y sú dané pozíciou v matici a na tejto pozícii je uložená hodnota z. Z týchto troch bodov následne vytvoríme rovnicu roviny.

Túto rovinu otestujeme na samotnej hĺbkovej mape, pričom si zapisujeme počet uzlov, ktoré do roviny patria. Pri zisťovaní prieniku roviny a bodu berieme do úvahy rčitý šum a nepresnosti, preto aj body ktoré sú veľmi blízko roviny zadefinujeme, že do roviny patria.

Tento postup opakujeme a vytvoríme viacero rovín, pričom pri každej novej rovine porovnáme súčet boodv, ktoré do roviny patria a uložíme tú, ktorá má viac bodov. Takto po niekoľkých opakovaniach získame rovnicu roviny, do ktorej patrí najväčší počet bodov.

Implementácia v programe

Algoritmus pre vytvorenie roviny je v programe implementovaný ako samostatná trieda, ktorá nekomunikuje s programom. V rámci ďalšej implementácie sme plánovali využiť túto rovinu pre vytvorenie obmedzenia grafu. Toto spôsobí zastavenie grafu v momente, keď sa stretne s rovinou.

3.3.2 Funkcionalita zoomu

Aktuálne prebieha vývoj približovania a vzdalovania vykresleného grafu. Pre získanie ruky sa používa zariadenie Kinect. Následne sa vyberie malý región okolo ruky a vykoná sa segmentácia ruky pomocou metódy *growing regions*. Získame vysegmentovanú ruku z ktorej pomocou metód pre hľadanie kontúr vieme vypočítať, či je ruka otvorená alebo uzavretá.

Tento modul sa má ďaľej rozšíriť o porovnanie počtu vystretých prstov a následne o približovanie alebo vzdaľovanie grafu podla vzdialenosti ruky. Pokiaľ má používateľ ruku vystretú, funkcionalita sa vykonáva. Keď používateľ ruku zavrie v päsť, približovanie (alebo vzdaľovanie) sa znefunkční a umožní tak používateľovi presunúť ruku na inú pozíciu.

3.4 Testovanie

Táto časť zahŕňa opis testovania našej aplikácie, ktorá prebiehala tak interne ako aj externe (testovaním tretej strany, ktorou sa stal ďalší tím, ktorý využíva Kinect zariadenie).

3.4.1 Testovanie treťou stranou

Po spolupráci s tímom č.7 Carlos, ktorý svoj vlastný projekt tiež rieši prostredníctvom Kinect zariadenia, sme vzájomne vykonali testovanie príslušných aplikácií. Dotazník, ktorý vyplnil tím Carlos ohľadom nášho vytvoreného produktu sa nachádza v tejto podkapitole.

Aký je Váš pocit po prvom zhliadnutí programu?
Na prvý pohľad program vyzerá zaujímavo, sme zvedaví akú funkcionalitu poskytuje.

Aké sú Vaše očakávania od programu?
Od programu očakávame možnosť manipulácie s grafovými dátami. Interakciu s ním v podobe otáčania, zväčšovania alebo zmenšovania grafu v priestore a iné druhy manipulácie, ktoré sú intuitívne vzhľadom pre osobu stojacu pred sklom, na ktoré sa premieta.

Mali ste možnosť sa s takouto aplikáciou stretnúť už predtým? Alebo s aplikáciou v podobnej forme?
S podobnou aplikáciou sme sa ešte nestretli. Existuje veľa aplikácii pre senzor Kinect, či už v hrách alebo inej forme, nie však pri obdobných aplikáciách, ktoré vyžadujú väčšiu presnosť.

Videli ste túto technológiú použitú niekde v praxi?
Žiaľ s touto technológiou sme sa nikde v praxi nestretli. Avšak veľakrát sme ju videli v rôznej podobe vo filmovéj tvorbe. Naopak, s ovládaním značkou sme sa už stretli pri tabletoch alebo smartfónoch.

Čo sa Vám páči pri vizualizácií?
Páči sa nám prehľadnosť grafov. Vďaka premietaniu na fóliu je zobrazovacia plocha o mnoho väčšia a lepšie sa v nej orientuje. Zaujímavé bolo zobrazenie webových stránok pomocou uzlov na takejto veľkej ploche.

Čo sa Vám páči na ovládaní?
Je to zaujímavé. Zatiaľ sme mali možnosť vidieť iba základnú funkcionalitu, ale vyzerá to

funkčne a použiteľne. Bolo by zaujímave, keby bolo možné okrem otáčana a označovania cez Kinect používať aj iné možnosti ovládania. Pozrieť si graf a otáčať si ho značkou pred sebou je asi ľahšie ako otáčať ho myšou.

Ktorá funkcionalita sa Vám najviac páčila?
Najlepšie sa nám pracovalo so zariadením Kinect cez pohyby rúk a zvukové gestá, ktoré zlepšili celkový dojem aplikácie.

Máte pocit, že Vás senzory alebo iné zariadenia ovplyvňujú, resp. že Vám prekážajú?
Senzory nie je problém po čase začať ignorovať. Pri interakcii veľmi blízko skla, sme si všimli presvietenie projektorom.

Je pochopenie ovládania zložité? Zmenili by ste niečo?
Ovládanie nie je intuitívne, ale dá sa pochopiť. Uvítali by sme viac možností ovládania grafu a jeho optimalizáciu. Pri testovaní manipulácie sa totiž v programe vyskytovali chyby v podobe zlej reakcie na používateľa.

Je vizualizácia údajov dostatočne plynulá?
Pri naozaj veľkých grafoch program chvíľu seká kým sa graf neusporiada. Občas tiež seká pri detekcii tváre a celkovo je program dosť náročný na výkon. Otáčanie grafu rukou je plynulé.

Objavili ste neočakávané stavy, neočakávané správanie programu?
Reakcie boli občas nepresné. Inak sme však nezaznamenali nežiadúce stavy programu. Pri pohybe grafu pomocou rúk cez zariadenie Kinect sme zaznamenali výrazné otočenie grafu keď sme sa s rukou dostali na kraj snímanej oblasti. Následne sme zistili, že toto je plánovaná funkcionalita a lahko sme sa naučili ju používať.

V čom vidíte nedostatky nášho produktu?
Myslíme si, že používateľské rozhranie je neprehľadné. Pri mnohých prvkoch nie je jasná ich funkcionalita. Pomohli by lepšie popisy jednotlivých tlačítok. Program je občas nestabilný a vyžaduje si ešte ďalšie úpravy.

Viete si predstaviť využitie tohto riešenia v praxi?
Toto riešenie je očividne prvotne myslené na výskumné účely ale dalo by sa využiť aj v

praxi, v ekonómii alebo v školách, tam kde sa používajú grafy.

Podarilo sa Vám pomocou funkcií ovládania dosiahnuť Vaše ciele?
Pri testovaní sme si stanovili základné ciele ako vyznačenie časti uzlov, manipuláciu s nimi ako aj pohyb grafov pomocou pohybov rúk a zvukových povelov. Podmienky sme boli schopní splniť ale celková práca nie je výrazne lepšia ako s použitím klasickej myšky či klávesnice. V tomto smere je na danom programe ešte čo vylepšovať a uvedomujeme si, že program je ešte vo fáze vývoja.

3.4.2 Testovacie scenáre a statické testovanie

Testovací scenár 1 – Spustenie a ukončenie aplikácie		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Spustiť súbor *3DVisual.exe*.	OK
2a	Stlačenie tlačidla *File* a následne výber *Quit*.	OK
2b	Stlačenie tlačidla *„close“*	OK

Testovací scenár 2– Načítanie vstupného súboru graphml		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Stlačenie tlačidla *File* a následne výber *Load graph from file*..	OK
2	Výber umiestnenia súboru s príponou *graphml.*	OK
3	Označenie súboru kliknutím na jeho názov. *“*	OK
4	Stlačenie tlačidla Ok	OK

Testovací scenár 3– Face recognition		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Stlačenie tlačidla *Start camera*	OK
2	Select Face recognition and press Start FaceRec	OK
3	Select camera device and press apply	OK
4	Point camera towards face. If camera strats to track face on screen test passes.	OK

Testovací scenár 4– Speech recognition		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Press start speech button	OK
2a	Say command “Select all nodes”	OK
2b	Say command “Select left side”	OK
2c	Say command “Select right side”	OK
2d	Say command “Unset restriction”	OK
2e	Say command “Sphere”	OK
3	If corresponding command is executed test has passed	OK

Testovací scenár 5– Kinect hand tracking (graph rotation)		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Select start Kinect button	OK
2	Select start button	OK
3	Stand in front of the Kinect and make hand gesture	OK

Testovací scenár 6 – Marker detection (mouse movement)		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Stlačenie tlačidla *Start camera*	OK
2	Select Marker detection and press Start Marker	OK
3	Select camera device and press apply	OK
4	Select change Markers to switch from graph rotation to mouse movement by graph	OK

Testovací scenár 7 - Marker detection		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Stlačenie tlačidla *Start camera*	OK
2	Select Marker detection and press Start Marker	OK
3	Select camera device and press apply	OK
4	Point camera towards marker. If camera strats to track maker on screen test passes.	OK

Testovací scenár 8 – Marker graph movement		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Stlačenie tlačidla *Start camera*	OK
2	Select Marker detection and press Start Marker	OK
3	Select camera device and press apply	OK
4	Point camera towards marker. If camera strats to track maker on screen test passes.	OK

Testovací scenár 9 – Background from camera		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Select edit button	OK
2	Select options	OK
3	Select Viewer/Skybox	OK
4	Select noise option	OK
5	Change option to “3”	OK
6	Select apply then close button	OK
7	Background is drawn from camera source	OK

Testovací scenár 10 – Hand gesture edge of screen rotation		
Tester: Duško Dogandžić	Date: 20.5.2014	
Číslo kroku:	Popis činnosti	Výsledok
1	Select start Kinect button	OK
2	Select start button	OK
3	Move hand to the edge of the screen	OK
4	Graph rotates seamlessly around its axis	OK

Static check report for Arvis Release 1.4

Warnings and errors reports still present. Some of the warning and are false postives.

[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::handZ' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::getArrayHands' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::isTwoHands' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::slidingHand_x' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::slidingHand_y' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::slidingHand_z' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::slidingHand_type' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::mDepthX' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::mDepthY' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectHandTracker.cpp:5]: (warning) Member variable 'KinectHandTracker::mDepthZ' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectThread.cpp:13]: (warning) Member variable 'KinectThread::kht' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\KinectThread.cpp:13]: (warning) Member variable 'KinectThread::mKinect' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Kinect\RansacSurface\PlaneAlgorithm.cpp:6]: (warning) Member variable 'PlaneAlgorithm::d' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Model\NodeDAO.cpp:348] -> [Arvis-1.4\Arvis-1.4\src\Model\NodeDAO.cpp:351]: (warning) Possible null pointer dereference: conn - otherwise it is redundant to check it against null.
[Arvis-1.4\Arvis-1.4\src\Model\NodeDAO.cpp:382] -> [Arvis-1.4\Arvis-1.4\src\Model\NodeDAO.cpp:387]: (warning) Possible null pointer dereference: conn - otherwise it is redundant to check it against null.
[Arvis-1.4\Arvis-1.4\src\OpenCV\CamSelectCore.cpp:15]: (warning) Redundant assignment of 'app' to itself.
[Arvis-1.4\Arvis-1.4\src\Speech\KinectSpeechThread.cpp:7]: (warning) Member variable 'KinectSpeechThread::isConnected' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::_delta_frame_time' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::_last_frame_time' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::decelerateSideRate' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::decelerateForwardRate' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::decelerateVerticalRate' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::deceleratePitchRate' is not initialized in the constructor.

[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::t0' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::dt' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::lastDistance' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::automaticMovementInitialized' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::t1' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::t2' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::w1' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::w2' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::targetDistance' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::cameraPositions' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\CameraManipulator.cpp:28]: (warning) Member variable 'CameraManipulator::targetPositions' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\MouseControl.cpp:6]: (warning) Member variable 'MouseControl::clickX' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\MouseControl.cpp:6]: (warning) Member variable 'MouseControl::clickY' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Viewer\RestrictionManipulatorsGroup.cpp:10]: (warning) Member variable 'RestrictionManipulatorsGroup::manipulatorCreator' is not initialized in the constructor.
[Arvis-1.4\Arvis-1.4\src\Model\GraphDAO.cpp:246]: (error) Uninitialized variable: newGraph
[Arvis-1.4\Arvis-1.4\src\Model\GraphDAO.cpp:249]: (error) Uninitialized variable: newGraph
[Arvis-1.4\Arvis-1.4\src\Viewer\ShapeVisitor_VisualizerCreator.cpp:53]: (error) Uninitialized variable: radius

Static Check statistics

Total static check statistic	
Number of files scanned	140
Errors	3
Warnings	65
Stylistic warnings	68
Portability warnings	42
Information messages	3

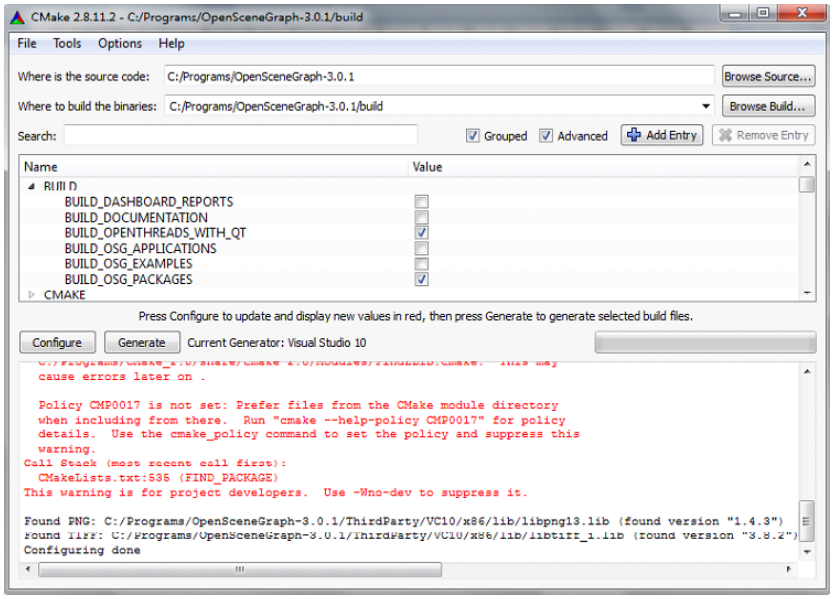
A Návod na inštaláciu

A.1 Návod na Windows

Tento návod bol úspešne otestovaný na operačnom systéme Windows 7 a 8. Na inštaláciu potrebujeme klonovať projekt Arvis z GitHubu [1], CMake 2.8.11, OpenSceneGraph (zdroje, ktoré budú buildované vo Visual Studio 2010), Kinect for Windows SDK, . Je dôležité mať nainštalované Microsoft Visual Studio 2010 SP1. Ďalej treba mať Qt 4.8.5 a keďže sa jedná o staršiu verziu, treba k tomu doinštalovať aj QtCreator. Pre pohodlnejšiu prácu s premennými prostrediami je tiež vhodné mať nainštalovaný program RapidEE. Taktiež je nutné mať pre inštaláciu knižnice 3rd party dependencies [2]. Postup na inštaláciu by mal vyzerať nasledovne:

1. Nainštalovať CMake. Cesta ku CMake ďalej v texte ako %CMAKE_DIR%.
2. Nainštalovať Qt a QtCreator. Cesta do Qt dalej ako %QT_DIR%.
3. Stiahnuť OSG - cestu do OSG ďalej v texte ako %OSG_DIR%.
4. Stiahnuť 3rd party knižnice, ideálne vytvoriť priečinok %OSG_DIR%\ThirdParty\.
5. Otvoríme si RapidEE pre zmenu enviroment variables.
   (a) do PATH pridáme nasledovné premenné:
      - %CMAKE_DIR%\bin
      - %QT_DIR%\4.8.5\bin
      - %OSG_DIR%\ThirdParty\VC10\x86\bin
   (b) vytvoríme variable CMAKE_INCLUDE_PATH a pridáme:
      - %OSG_DIR%\include
      - %OSG_DIR%\ThirdParty\VC10\x86\include
   (c) vytvoríme variable CMAKE_LIBRARY_PATH a pridáme:
      - %OSG_DIR%\ThirdParty\VC10\x86\lib.
6. Ak ste stiahli OSG binary files preskočte na krok 10. Spustíme Cmake-gui. Ako source priečinok si nastavíme %OSG_DIR% a build cestu %OSG_DIR%\build. Spustíme configure a vyberieme Visual Studio 2010.

[1] Zdroj: https://github.com/marconak/Arvis, dostupné z webu v máji 2014.
[2] Zdroj: http://ulozto.sk/xaZJArtg/vc10-zip, dostupné z webu v novembri 2013.



Obr. 20: *Ukážka CMake*

7. Configure by vám mal prebehnúť v poriadku a nájsť aj PNG a iné knižnice, čo potrebuje. Odporúčam zaškrtnúť grouped a advanced a v BUILD záložke zaškrtnúť podľa obrázku, nech nebuildujete zbytočne example aplikácie a podobne.

8. Stlačíme znova configure, ak nie sú žiadne problémy následne stlačíme generate.

9. V priečinku %OSG_DIR%\build spustíme .sln pre visual a dáme build project (vytvoria sa knižnice a .dll). Môžeme vytvoriť pre Debug aj pre Release (cca. 10 minút).

10. Znova pridáme enviroment variables.

    (a) do PATH pridáme:

    - %OSG_DIR%\build\bin,

    (b) do CMAKE_LIBRARY_PATH pridáme:

    - %OSG_DIR%\build\lib.

11. V priečinku %Arvis% vytvoríme _build a _install.

12. Spustíme QtCreator. File → Open file or project. Vyberieme CMakeLists.txt z %Arvis%.

13. Ako build directory zvolíme %Arvis%_build

14. Arguments: -DCMAKE_BUILD_TYPE-
=Debug alebo -DCMAKE_BUILD_TYPE=Release a Generator NMake pre verziu 4.8.5. V prípade, že nenašlo generator postupujeme nasledovne:

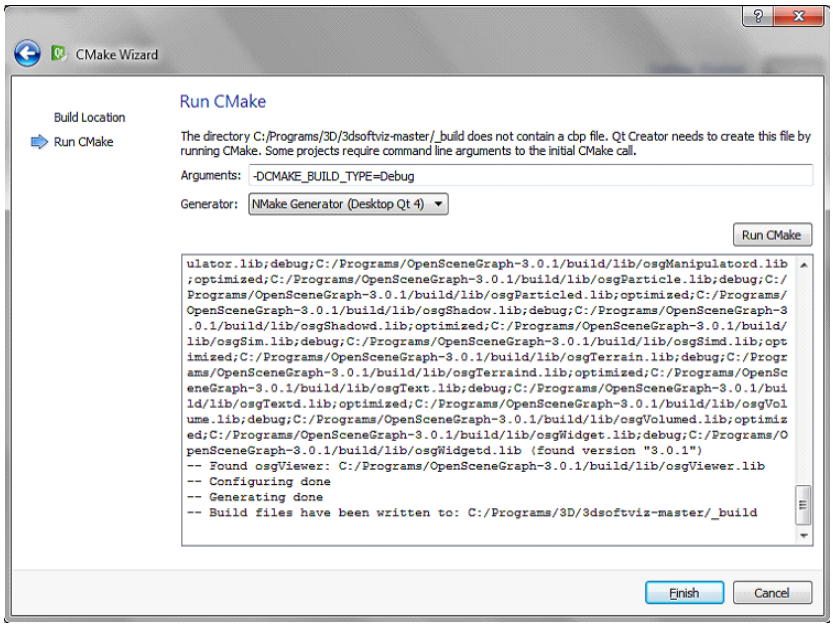
    (a) Tools → Options → CMake a ako cestu nastavíme %CMAKE_DIR%-\bin\cmake.exe,

    (b) V Kits by sme mali mať nastavené Qt verziu 4.8.5 a compiler VS 2010,

    (c) Vo Qt versions ak nenašlo cestu do %QT_DIR%\4.8.5\bin\qmake.exe , treba ju nastaviť,

    (d) teraz by vám už malo nájsť generator.

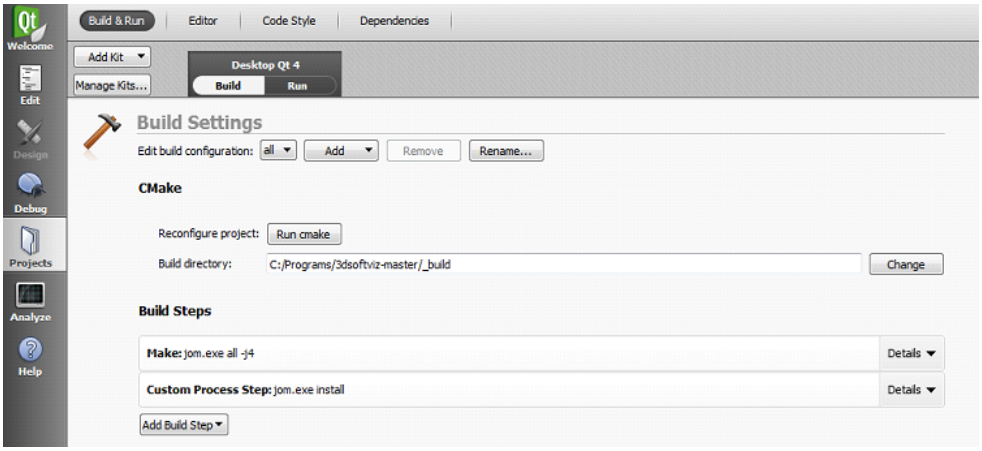
15. Spustime Run Cmake.



Obr. 21: *Spustenie CMake*

16. Po dokončení v ľavej lište vyberieme Projects → build & run → build a v Build steps nastavíme ku Make: *jom.exe* additional arguments *-j4*.

17. Pridáme k tomu ešte Add build step → Custom process step. command: *jom.exe* a argument *install*.



Obr. 22: *Spustenie CMake*

18. Následne celý program zbuildujeme (kladivko vľavo dole).

19. Mali by ste mať uz program v _install/bin a mal by sa dať spustiť. Môžeme proces buildovania zopakovať aj pre Release/Debug

Inštalácia Kinect a rozpoznávanie hlasu

Pre úspešné inicializovanie Kinect zariadenia je potrebné mať vo Windowse nainštalovaný SDK, ktorý poskytuje Microsoft [3]. V prípade pridania funkcionality na rozpoznávanie hlasu musí mať používateľ nainštalovaný ešte Speech SDK [4] a tiež nastaviť systémovú premennú s názvom *KINECTSPEECH_DIR*, ktorá definuje cestu (priečinok), kde sa Speech SDK nainštaloval.

Inštalácia ďalších potrebných Kinect knižníc

NiTe2 a OpenNI2 sú ďalšie externé knižnice (prvá menovaná slúži primárne na detegovanie ruky a OpenNI2 na inicializáciu zariadenia), ktoré projekt používa a ich import je už vyriešený v rámci projektu. Potrebné je však modifikovať enviromentálne premenné $CMAKE\_INCLUDE\_PATH$ a $CMAKE\_LIBRARY\_PATH$ tak, aby si projekt našiel cestu k potrebným hlavičkovým a knižničným súborom. Postup by mal byť nasledovný, keď predpokladáme, že cesty ku knižniciam sú $\%$*NITE2_DIR*$\%$ a $\%$*OPENNI2_DIR*$\%$:

[3] Zdroj: http://www.microsoft.com/en-us/kinectforwindowsdev/start.aspx, dostupné z webu v máji 2014.
[4] Zdroj: http://www.microsoft.com/en-us/download/details.aspx?id=10121, dostupné z webu v máji 2014.

1. do premennej $CMAKE\_INCLUDE\_PATH$ nastavíme cesty $\%$*NITE2_DIR*$\%$\Include a $\%$*OPENNI2_DIR*$\%$\Include

2. do premennej $CMAKE\_LIBRARY\_PATH$ nastavíme cesty $\%$*NITE2_DIR*$\%$\Lib a $\%$*OPENNI2_DIR*$\%$\Lib

3. kvôli ďalším *.dll* súborom je potrebné do $CMAKE\_INCLUDE\_PATH$ nastaviť ešte cestu $\%$*NITE2_DIR*$\%$\Samples\Bin\OpenNI2\Drivers

Ak sa v rámci spustenia *cmake* vypíše *OpenNI2 FOUND* a *NITE2 FOUND*, tak prebehla inštalácia knižníc a nastavenie ciest pre projekt úspešne. Táto funkcionalita je obmedzená na platformu Windows. Program ošetruje problém, ak Kinect senzor nemáme k dispozcíii, alebo nemáme nainštalované a nastavené správne knižnice. Je vhodné najprv po zmenených nastaveniach reštartovať systém (alebo minimálne QtCreator a vymazať build aj install adresáre, aby sme následne spustili hneď *cmake*). Je nutné poznamenať, že projekt sa skompiluje aj v prípade, že nemáme zapezpečenú podporu Kinect zariadenia. Žial, počas vývoja tohto projektu došlo k zaniknutiu oficálnej stránky OpenNI a teraz je potrebné nainštalovať si knižnice z GitHub stránky projektu (alebo kontaktovať tím, vid. nižšie).

Inštalácia ďalších knižníc

Okrem spomenutých knižníc v návode je potrebné cez git konzolu po klonovaní repozitára spustiť príkaz *submodule*, aby pribudol *libnoise*. Taktiež do projektu pribudla knižnica ArUco (potrebné inštalovať), ten má problémy na Windowse s OpenGL. Riešením je zakázať build v ArUco GL príkladoch, čo je možné nastaviť v $dependencies/aruco/utils$ v *CmakeLists* projektu. Všetky dôležité informácie s opisom možných problémov pri spustení aplikácie sú riešené aj na wiki stránke tímového Redmine [5] - ak by platnosť stránky vypršala, treba kontaktovať tím na mail *teamtp05@gmail.com* alebo *soos.dano@gmail.com*. Rovnako treba spraviť, ak chce mať používateľ nainštalovanú aplikáciu na operačnom systéme Linux, avšak na tejto platforme je momentálne funkcionalita obmedzená (projekt bez Kinectu).

OpenCV má vlastný *.cmake* súbor vo svojom *build* adresári, stačí si túto cestu nastaviť ako systémovú premennú menom $OpenCV\_DIR$ (cesta má viesť k build adresáru nainštalovaného OpenCV). Odporúčané je mať nainštalovanú verziu 2.4.6 [6].

[5] Zdroj: http://team05-13.ucebne.fiit.stuba.sk:443/projects/letny-semester/wiki, dostupné z webu v máji 2014.

[6] Zdroj: http://opencv.org/downloads.html, dostupné z webu v máji 2014.

Časté problémy

Ak neboli nájdené submoduly *libnoise* alebo *aruco*, treba v projekte spustiť príkazy:
git submodule init,
git submodule update.

V prípade kompilačných chýb bude pravdepodobne nutné nahradenie niektorých súborov:

- po pridaní aruca pri kompilácii, ak chyba obsahuje, že nebol nájdený *glut.h*, tak to je súbor *dependencies/aruco/utils/Cmakelist.txt*[7],
- veľa malých kompilačných chýb, ktoré obsahujú *ambiguous uint*, tak
- problém so súborom *dependencies/libnoise/src/basictypes.h*[8] - niekedy je dobré po zmene spustiť *clean* alebo rovno vymazať celý *_build*.

V prípade odstránenia pomocných výpisov Aruca treba zakomentovať v */dependencies/aruco/src/marker.cpp* v metóde *calculateExtrinsics()* 297. riadok *cout«(*this)«endl;*.
Pre nastavenie databázy sú potrebné nasledujúce kroky:

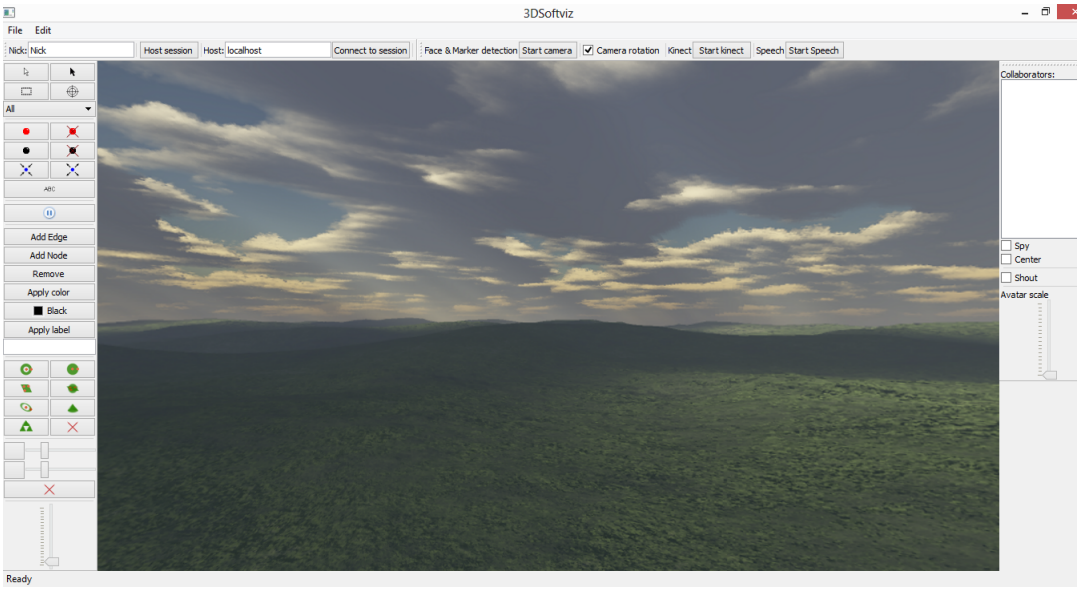
- nainštalovať Postgre DB, a vytvoriť prázdnu databázu;
- spustiť sql skript v projekte: tmp/tp_db_initialization.sql;
- v konfiguračnom súbore upraviť tieto 4 riadky:
  1. Model.DB.DbName=arvisDB,
  2. Model.DB.HostName=localhost,
  3. Model.DB.Pass=tajne heslo,
  4. Model.DB.UserName=postgres.

Vývoj knižníc a oficálne stránky OpenNI2 a NiTE2 boli počas trvania tohto projektu pozastavené, ich inštalácie sú však naďalej k dispozícii na webe[9].

[7] Zdroj: http://ulozto.net/xUzixWNX/cmakelists-txt, dostupné z webu v máji 2014.
[8] Zdroj: http://ulozto.net/xY25AQaF/basictypes-h, dostupné z webu v máji 2014.
[9] Zdroj: http://uloz.to/xpz9zqAi/openni-nitte-rar, dostupné z webu v máji 2014.

B Používateľská príručka

Aplikácia 3DSoftviz sa spúšťa cez súbor *3DSoftviz.exe*. Na Obrázku 23 vidíme používateľské rozhranie aplikácie.



Obr. 23: *Používateľské rozhranie aplikácie*

Základná funkcionalita

V ľavom hornom rohu sa nachádza hlavné menu aplikácie, v ktorom sa dá načítať súbor (preddefinovaný je súbor s príponou *graphml*) z adresára. Takisto sa dá uložiť graf a v sekcii *Edit* je možné zmeniť nastavenia programu.

Na ľavej strane je taktiež aj nástrojová lišta. Opciami *All*, *Node* a *Edge* je možné vybrať si, či chceme selektovať hrany spolu s uzlami, alebo len uzly, resp. hrany. Predvolene je program nastavený na *No-select mode* (biela šípka), čiernou šípkou sa môžeme prepnúť na stav *Single-select mode*, čo nám umožňuje sústredenie sa na práve jeden objekt - môže to byť hrana aj uzol. Tretia možnosť v menu je *Multi-select mode*, čím sa dá vybrať v trojrozmernom zobrazení viacero objektov naraz. Štvrtá opcia v tejto rubrike zobrazí centrálny pohľad na práve vybranú časť trojdimenzionálneho grafu. Pokiaľ sme nastavení na *No-select mode*, vieme hýbať vybraným uzlom.

Pod týmto menu sú tlačidlá na pridávanie meta-uzlov a na zmazanie meta-uzlov, pričom, pochopiteľne, zmazať sa dajú len tie popridávané meta-uzly. Existuje aj možnosť zafixovania a odfixovania vybraných uzlov. Možnosťou *Merge nodes together* sa naskytuje

možnosť zlúčiť vybrané uzly do jedného spoločného uzla. Takýto uzol sa bude v pokračovaní zobrazovať s modrou farbou. Týmto spôsobom je ľahké zrušiť zlúčenie týchto uzlov možnosťou *Separate merged nodes*.

Ak hýbeme uzlami, často sa formuje aj celý graf - túto akciu môžeme vypnúť tlačidlom *Play*.

Opcia *Add Edge* pridáva hranu medzi dvoma vybranými uzlami, kde ešte nie je hrana, inak končí akcia chybovou hláškou. Ideálne je čiernou šípkou vybrať jeden uzol a bielou šípkou ho nastaviť na také miesto, kde sa ho dá spojiť s druhým uzlom - je potrebné mať nastavenie *Node* v takomto prípade spolu s *Multi-select mode*. *Remove* slúži na zmazanie uzlov alebo hrán. Ak sa rozhodneme pre zmazanie hrany, uzly prepojené s touto hranou na grafe ostávajú.

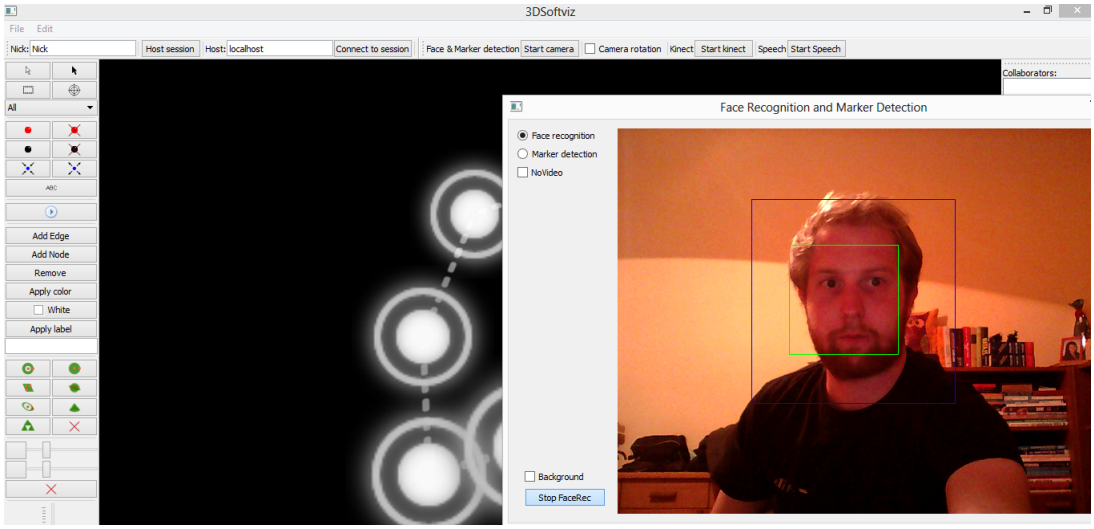
Aplikovanie rôznych farieb sa uskutočňuje spôsobom, že najprv sa spraví operácia, ktorú chceme znázorniť farebne (napr. výber jednej podčasti grafu) a až potom sa na to aplikuje farba. Textové pole sa používa tým istým spôsobom ako sfarbenie grafu. Zapnúť tieto texty treba tlačidlom, ktoré sa nachádza nad možnosťou *Play*. Ďalšie štyri možnosti tvoria aplikácie priestorových ohraničení (povrch gule, obsah gule a rovina), použité a vybrané ohraničenia je možné aj odstrániť.

Ak je potreba zväčšiť aplikačnú časť programu, je možné kliknúť na ľubovolné miesto hlavného panelu pravým klávesom, a vypnúť niečo z trojice *Tools* (nástroje), *Collaboration* (kolaborácie) alebo *Network* (pripojenie).

Pridaná funkcionalita v rámci projektu ARVis

V hornom menu sa nachádzajú tlačidlá *Start Camera*, *Start Kinect* a *Start Speech*, takisto ako opcia *Camera Rotation*. *Camera Rotation* definuje spôsob rotácie a pohybu grafu - ak je nastavený tento stav, tak sa pohybuje kamera nasmerovaná na graf na základe pohybu tváre, značky alebo rúk. V druhom prípade sa rotuje na základe našich akcií samotný graf.

Aktivovaním *Start Camera* sa nám zjaví ďalšie okno, v ktorom treba zvoliť, či chceme využívať funkcionalitu rozpoznávania tváre alebo značky. Ak sme sa rozhodli pre prvú možnosť, kliknutím na *Start FaceRec* máme možnosť zvoliť kamerové zariadenie a následným potvrdením sa nám v okne objaví záber z našej kamery - v prípade detegovanej tváre (detekcia je reprezentovaná zeleným obdĺžnikom) sa kamera alebo graf pohybuje vďaka pohybu tváre. Kliknutím na *Background* je možné nastaviť aktuálne snímanie ako pozadie pre graf. Obrázok 24 znázorňuje stav aplikácie pre prípad použitia, keď sa hľadá tvár z obrazu kamery.



Obr. 24: *Používateľské rozhranie aplikácie - okno s obrazom kamery pre detekciu tváre*

V prípade, ak používateľovi vadí pridané okno, môže ho zatvoriť po tom, ako sa detegovala tvár a môže ďalej pokračovať s pohybom grafu alebo kamery. Ukončiť túto akciu je možné tlačidlom *StopFaceRec* (ak používateľ zatvoril okno, môže ho vrátiť na grafický interfejs opätovným kliknutím na *StartCamera* a potom pozastaviť detekciu).

Kliknutím na *Marker detection* sa ľavá strana okna prispôsobí pre ovládanie funkcionality rozpoznávania značky. Po stlačení tlačidla *Start Marker* sa zobrazí okno pre výber a nastavenie kamerového zariadenia. Po výbere sa v okne vpravo zobrazí výstup z kamery určenej pre rozpoznávanie značky a graf sa začne otáčať a pohybovať so značkou.

V ľavom menu sa tiež nachádza niekoľko nastavení. Predvolene sa graf pohybuje podľa značky, ako keby sa kamera pozerala na používateľa. Opciou *Marker is behind* sa dá prepínať medzi opačným stavom.

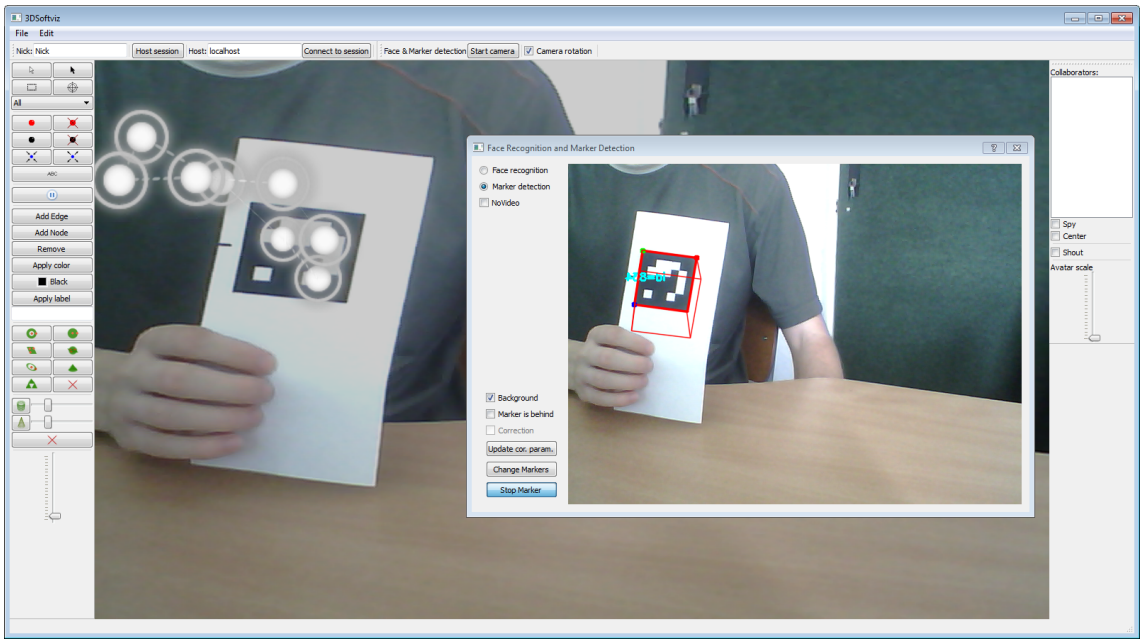
Podľa predvolených nastavení sa značka pohybuje tak ako keby sa kamera pozerala vo vodorovnom smere. Ak by sa pozerala napr. na stôl pod miernym sklonom dole, graf by sa pri posúvaní značky po stole neposúval korektne. Preto je možné nastaviť tlačidlo *Update cor. param.* a nastaviť korekčné parametre. Najskôr je potrebné nastaviť značku do polohy kedy je detegovaná na spodnom okraji a následne stačiť toto tlačidlo. Po nastavení sa aktivuje opcia *Correction*, ktorou je možné zapnúť korekciu.

Značkou je tiež možné ovládať aj kurzor, pričom je to možné vykonávať súčasne s použitím druhej značky. Ak má používateľ k dispozícií len jednu značku, môže tlačidlom *Change Markers* meniť spôsob jej použitia.

Medzi oboma funkcionalitami rozpoznávania tváre a značky je možné kedykoľvek sa prepínať, pričom sa v vpravo zobrazuje video pre práve aktuálnu voľbu. Opciou *NoVideo*

je možné vypnúť zobrazenie videa. Toto prepínanie a vypnutie zobrazenia video má vlyp len na zobrazenie len v rámci okna tohto ovládacieho okna *Face Recognition and Marker Detection* a neovplyvňuje to ani voľbu kamery pre video pozadie.

Funkciu video pozadia, pri ktorej sa v hlavnom okne zobrazuje video z kamery je potrebné najskôr umožniť zmenou parametra *Viewer.SkyBox.Noise* v konfiguračnom súbore na hodnotu 2 alebo 3 (odporúčané je 3). Video pozadie sa dá následne zapnúť opciou *Background* pri ktorúkoľvek kameru aktívnu pri rozpoznávaní tváre alebo značky, až do ich ukončenia. Touto opciou je možné prepínať medzi oboma zdrojmi video pozadia alebo ho aj vypnúť.



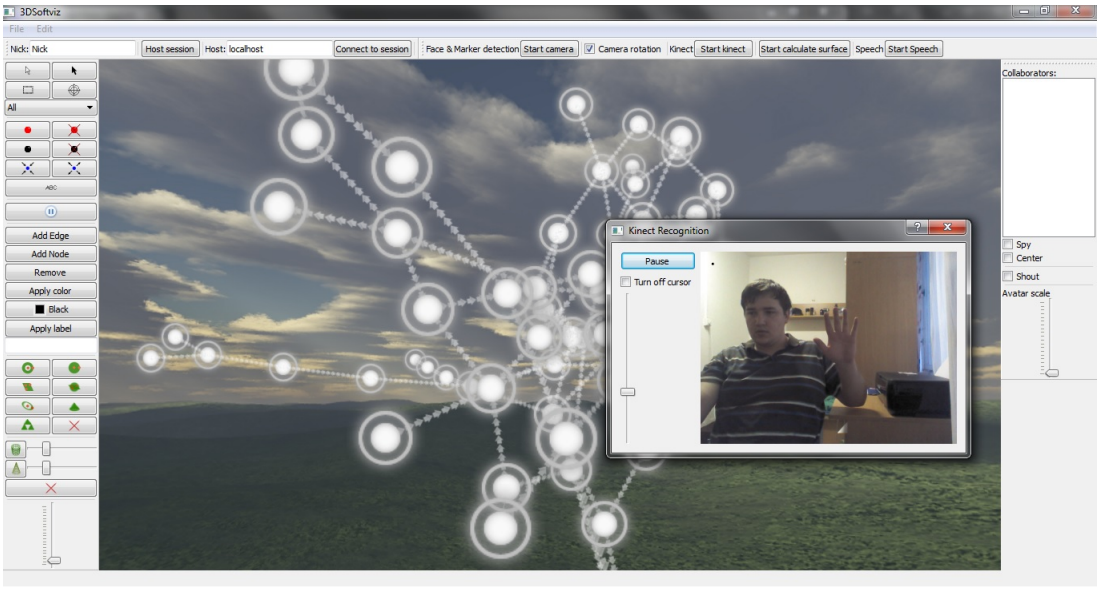
Obr. 25: *Používateľské rozhranie aplikácie - okno pre ovládenie rozpoznávanie značky s aktívnym video pozadím*

Start Kinect otvorí nové okno a po zapnutí kamery sa snaží program detegovať ruku používateľa. V prípade úspechu je možné rotovať grafom alebo kamerou (v závislosti od *Camera Rotation* ako už bolo spomenuté). Ak je *Turn off cursor* zaškrtnuté, tak v prípade akcie "kliku"(detekcia ruky spôsobom, keď rukou pohybujeme do hĺbky, nie vertikálne alebo horizontálne) sa aktivuje myš na spôsob, že reaguje na pohyb ruky. Ak táto možnosť nie je zapnutá, tak sa deteguje ruka a týmto pohybom manipuluje používateľ s grafom alebo s kamerou nasmerovanou na graf, v podobe rotovania. Ak sa používateľ dostane na kraj definovaného regiónu, tak sa graf začne automaticky otáčať až kým používateľ nespraví pohyb rukou do opačného smeru (tento prípad použitia platí aj pre rozpoznávanie tváre). Na Obrázku 26 je znázornená táto časť funkcionality. Zatvárať okno a zastaviť

snímanie je možné rovnakým spôsobom ako pri rozpoznávaní tváre.

Samozrejme, Kinect zariadenie je potrebné pre túto funcionalitu - takisto ako pre využitie tlačidla *Start Speech*. Hlasové povely, na ktoré reaguje program po kliknutí na toto tlačidlo sú nasledovné:

- *select all nodes* - vybratie všetkých uzlov,
- *select left side* - vybratie uzlov na ľavej strany,
- *select right side* - vybratie uzlov na pravej strany,
- *clear screen* - zrušenie vybratia uzlov,
- *sphere* - sformovanie gule pre vybrané uzly,
- *unset restrictions* - návrat k pôvodnému stavu; zrušenie akcie "sphere".



Obr. 26: *Používateľské rozhranie aplikácie - využívanie Kinectu pre detekciu rúk*

Aplikácia umožňuje fullscreen režím, ktorý sa aktivuje stlačením tlačidla [l] a tiež je možné schovať alebo zobraziť všetky toolbary, buď pomocou kontextového menu alebo tlačidlo [t]. Pre používanie týchto skratiek, je potrebné kliknúť najskôr na zobrazovaciu plochu.