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 I
Kontakt: teamtp05@gmail.com
Akademický rok: 20213/2014, zimný 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. Je v ňom obsiahnutý celkový pohľad na projekt spolu s cieľmi projektu, takisto ako dokumentácia k jednotlivým šprintom, ktoré sa v projekte udiali.

1.1 Celkový pohľad na projekt

Naším cieľom je rozšíriť funkčnosť predchádzajúcej verzie aplikácie o možnosť zobrazovať grafy v obohatenej realite, priniesť tiež nové spôsoby interakcie s nimi, a urobiť tak vizualizáciu oveľa zaujímavejšou.

Základná koncepcia riešenia:

- Spojenie doterajších verzií programu a vytvorenie verzie integrujúcej všetky doposiaľ pridané funkcie.
- Pridanie funkcie snímania a automatického rozpoznania polohy tváre používateľa pomocou knižnice OpenCV alebo Kinectu, aby sa graf mohol otáčať podľa jej polohy.
- Zmena virtuálneho priestoru za reálnu scénu. Používateľ sa môže pozerať cez poloprieswitné plátno do reálneho prostredia, napr. na jeho pracovný stôl, a na tomto sa mu bude nachádzať graf, ktorý zobrazuje.

  Umiestnenie grafu do priestoru sa dosiahne najskôr s použitím značky rozpoznávanej kamerou a knižnicou OsgArt, neskôr pomocou Kinectu. Kinectom sa získavajú informácie o tvare prostredia a z hĺbkovej mapy je vytvorený model prostredia. Tento potom dokážeme spracovať a získať vhodnú polohu pre umiestnenie grafu, napr. plochu stola, a obmedzenia pre rozmiestnenie grafu, napr. kde sú steny alebo iné objekty, pretože graf nemôže prechádzať cez prekážky. Spolu s reakciami grafu na pozíciu tváre sa dosiahne pocit, že virtuálny graf patrí do prostredia.
- Pridaním možnosti manipulovať s grafom pomocou gest sa otvárajú nové cesty k interakcii s grafom. Toto sa dosiahne použitím Kinectu.

- Tieto nové funkcie sa pridávajú ako voliteľné rozšírenie aplikácie, aby boli zachované aj pôvodné funkcie a ovládanie aplikácie pre používateľov, ktorí nemajú k dispozícii potrebné vybavenie.
- Naďalej je zachovaná multiplatformovosť aplikácie a pôvodný návrh architektúry s urýchlením procesu kompilovania vyčlenením niektorých stálych aj budúcich modulov ako submodulov vo forme knižníc.

2 Šprint 1

2.1 Analýza potrebných knižníc pre prácu na projekte

2.1.1 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.

2.1.2 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.

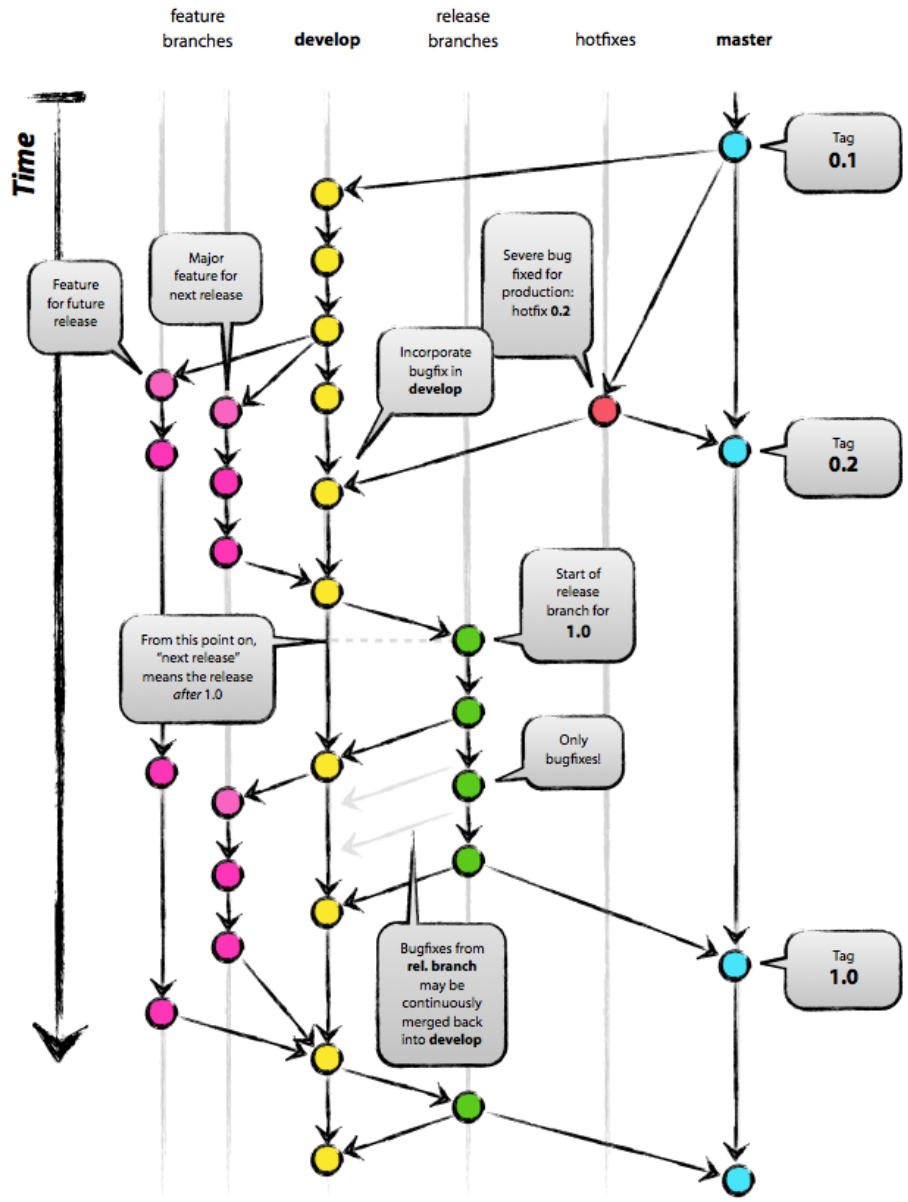
Pre potreby aplikácie vizualizácie dát v obohatenej realite je možné využiť openCV knižnicu pre rozpoznávanie tváre, ktorej poloha sa môže využiť pri vykresľovaní dát. Algoritmus pre rozpoznávanie tváre prebieha pomocou detekcie Haarových príznakov. Súčasťou openCV sú k dispozícii aj natrénované príznaky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

2.2 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]

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

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

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:

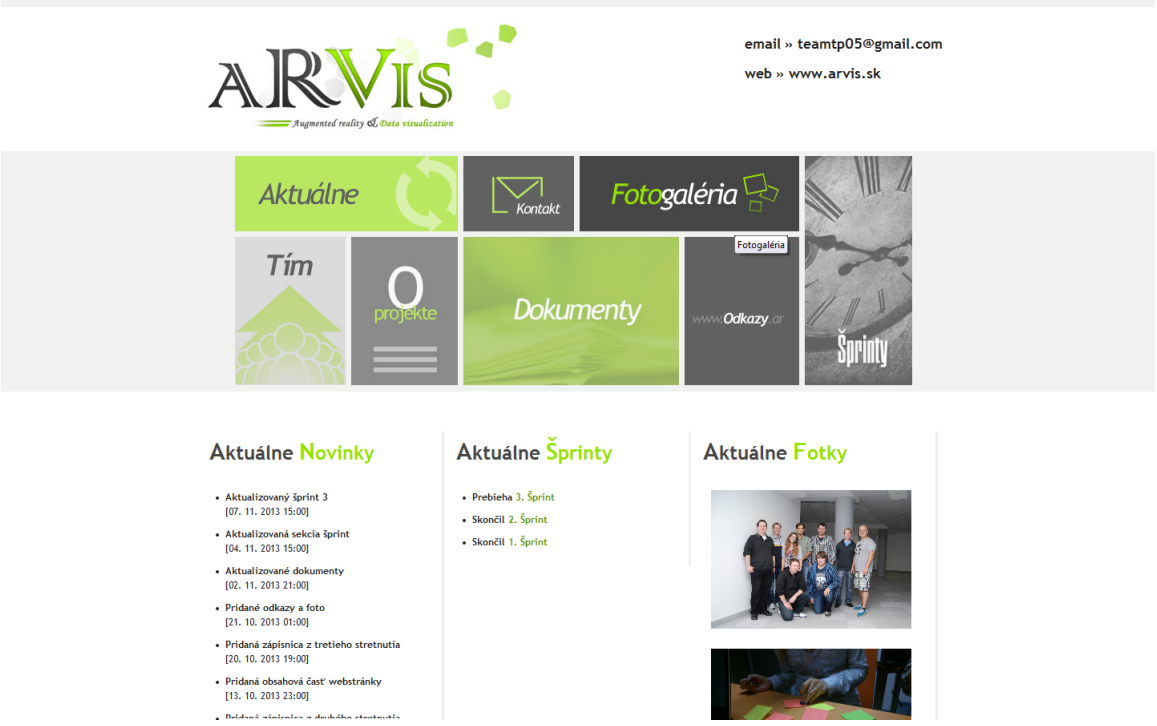
- *features* – tieto vetvy sú vytvárané pre novú funkcionalitu pre nasledujúcu, poprípade budúcu produkciu. Tieto vetvy vychádzajú priamo z *develop* vetvy, a v žiadnom prípade nemôžu integrovať s *master* vetvou. Táto vetva existuje počas vývoja funkcionality a po dokončení je spojená s *develop* vetvou 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.

2.3 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'*

3 Šprint 2

3.1 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`

3.2 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.

3.3 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.

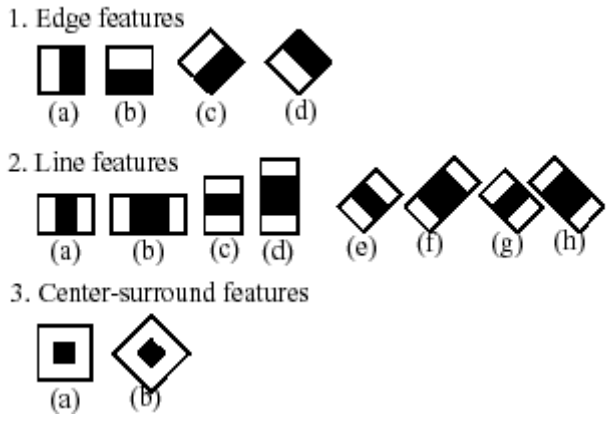
- Kompilátor mal problém s použitím knižníc Qt/qstring.h a Qt/qstringlist.h. Pre vyriešenie tohto problému bol pridaný ifdef blok, ktorý v prípade kompilácie na platformách apple používa knižnice qstring.h a qstringlist.h namiesto hore spomynaných. Zmena bola nutná v súboroch:
  - include/QOSG/CoreWindow.h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

3.4 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.

3.5 Rozpoznanie tváre

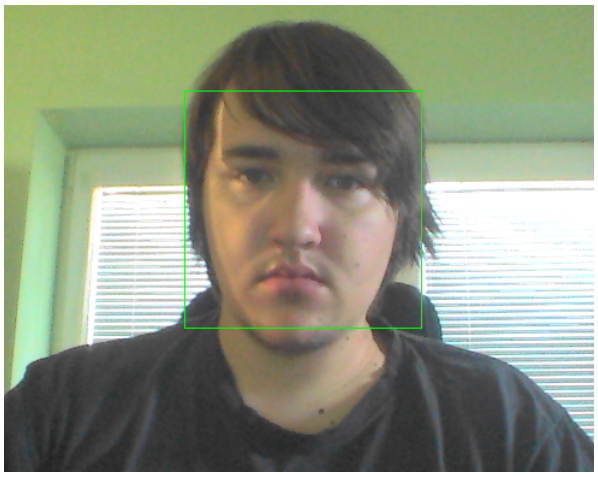
Pre funkciu rozpoznania tváre ako prvú funkcionalitu sme sa rozhodli použiť knižnicu openCV. V tejto knižnici sa nachádza súbor obsahujúci haar-like príznaky pre detekciu rôznych častí tela. Pre naše potreby sme využili súbor *haarcascade_frontalface_alt.xml* s naučenými príznakmi pre detekciu tváre.



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*

3.6 Aktualizácia 3DVisual o funkcionalitu z iných vetiev na GitHube

3.6.1 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.

3.6.2 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.

3.6.3 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.

3.6.4 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.

4 Šprint 3

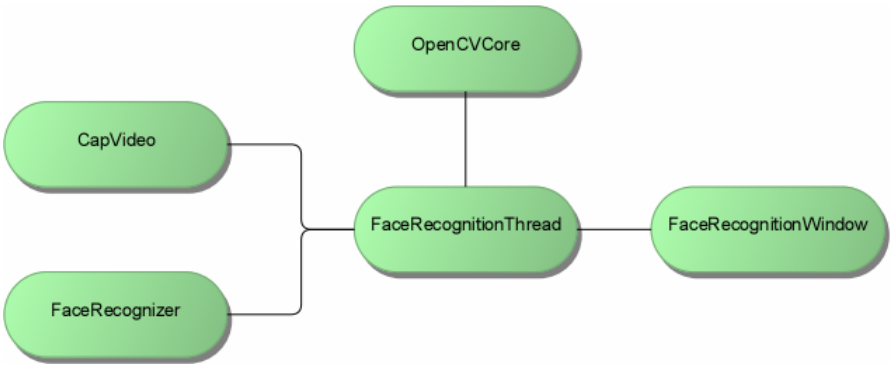
4.1 Import OpenCV

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.

4.2 Modularita v projekte

4.2.1 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.

4.2.2 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.

4.3 Úprava cmake súborov pre submodul Libnoise

4.3.1 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.

4.3.2 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.

4.3.3 Zhodnotenie

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

4.4 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.

4.4.1 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.

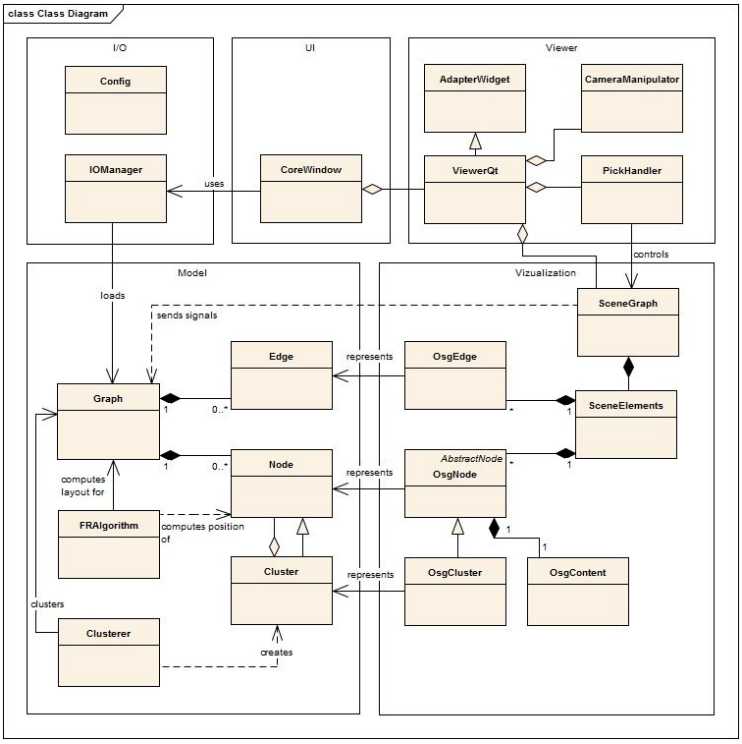
- Pohybovať grafom a aj kamerou v 3D ohraničenom priestore okolo zobrazeného grafu.
- Vo vykreslenom grafe umožňuje výber jedného alebo viacerých uzlov a hrán, ktoré je možné ďalej presúvať, meniť im farby.
- Možnosť zvoliť uzol alebo skupinu uzlov ako stred grafu okolo ktorého sa otáča kamera
- Vyberať 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.

4.4.2 Súčasná architektúra systému

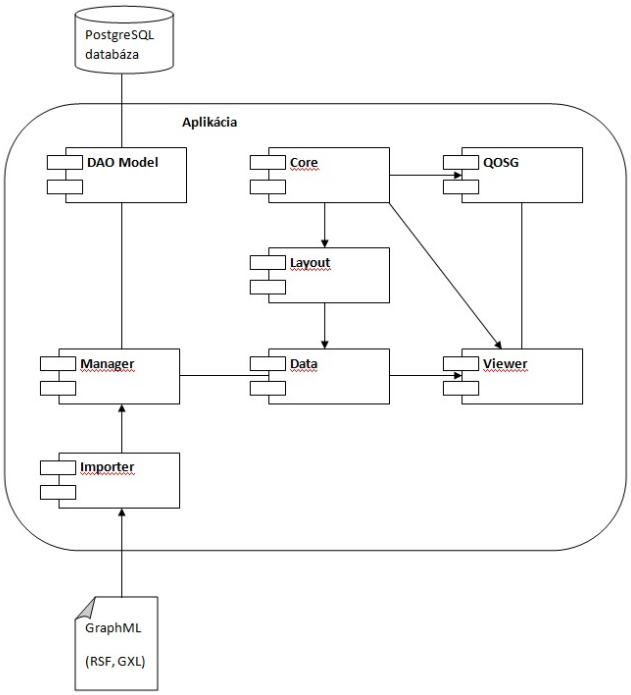
V aktuálnom stave je aplikácia rozdelená do niekoľkých modulov, podľa ich funkcie. Obr. 7 zobrazuje pôvodnú architektúru systému. Pre lepšie vysvetlenie obr. 6 zobrazuje podobnú štruktúru avšak s pohľadu hlavných tried.

- **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*

4.5 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.

5 Šprint 4

5.1 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.

5.1.1 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.

5.1.2 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.

5.1.3 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'.

5.2 Pridanie slotu pre otáčanie grafu podľa pozície tváre

5.2.1 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.

5.2.2 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.

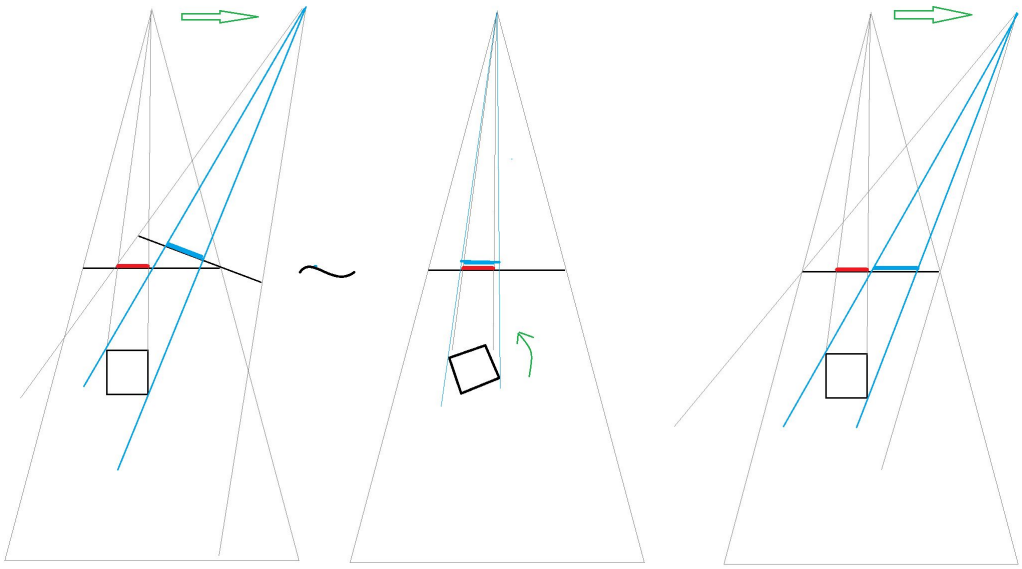
Kvôli zachovaniu pôvodnej funkcionality sme sa rozhodli pri otočení nemodifikovať priamo tento quaternion, ale pridávame nový quaternion $\_rotationHead$, ktorý bude vyjadrovať dodatočné otočenie podľa pozície tváre. Jeho otočenie je vypočítane tiež funkciou $trackball\,()$, avšak osobitne v horizontálnom 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.

5.2.3 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*

5.3 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.

5.3.1 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.

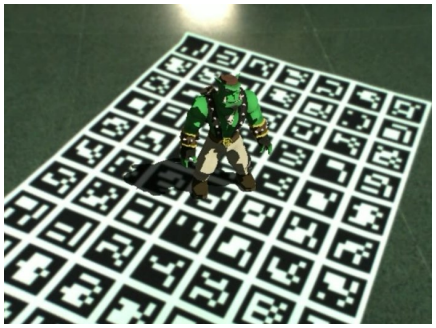
Ďalšou časťou úlohy bola úprava jednotlivých súborov a ich závislosti medzi nimi z hľadiska použitých *include* v nich. V jednotlivých súboroch boli odstránené nepoužívané *include* príkazy a skúmaná závislosť medzi ďalšími časťami.

5.4 ArUco - prieskumné prototypovanie

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

5.4.1 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,
- 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.

5.4.2 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

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

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.

5.5 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.

6 Šprint 5

6.1 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.

6.1.1 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.

6.1.2 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.

6.1.3 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.

6.2 Projekcia podľa vzdialenosti

6.2.1 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.

6.2.2 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.

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

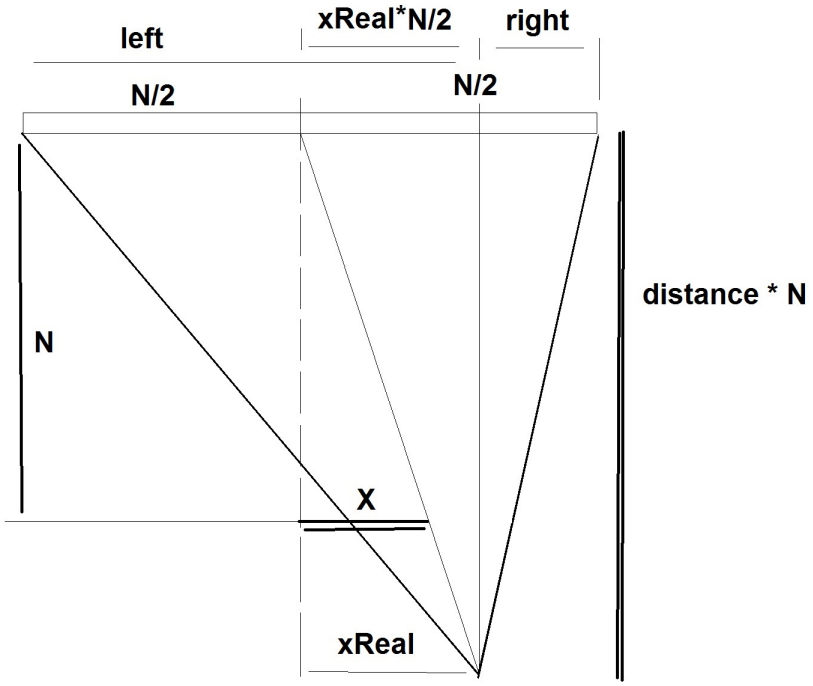
$$left = -(zNear - right) = -zNear + right$$

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.

6.2.3 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*

6.3 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.

6.3.1 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*.

6.4 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.

6.4.1 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.

6.5 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.

6.6 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.