Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ilkovičova2, 842 16 Bratislava 4

AugReality

Dokumentácia projektu-Inžinierske dielo

Vedúci tímu: Ing. Kapec Peter Phd.

Členovia tímu: Bc. Filip Škultéty,

Bc. Peter Belai,

Bc. Martin Mokrý,

Bc. Lukáš Hagara,

Bc. Patrik Berger,

Bc. Marek Roštár,

Bc. Michal Galbavý

Akademický rok: 2016/2017

Obsah

1. Úvod

Dokument inžinierske dielo opisuje technickú dokumentáciu projektu Vizualizácia informácií v obohatenej realite. Opisuje architektúru a postup rozširovania projektu. Základná architektúra je prebratá z minuloročných projektov.

V druhej kapitole sú opísané globálne ciele pre zimný semester, ktoré sa budeme snažiť splniť počas prvých 5 šprintov. Ciele sú rôznorodé, napríklad opravovanie warningov, zobrazenie modelu dlaní v scéne alebo vytvorenie datasetu zo zariadenia kinect 2. V tretej kapitole sa nachádza celkový obraz systému, ktorý pomocou UML diagramov opisuje systém. V 4. kapitole sú bližšie opísané moduly systému, konkrétne modul LeapAR, modul Kinect 2.

1.1. Podiel práce na dokumentácií k inžinierskemu dielu

Kapitola	Autor
Úvod	Filip Škultéty
Globálne ciele pre ZS, LS	Celý tím
Celkový pohľad na systém	Martin Mokrý
Celkový pohľad na systém: Kinect 2	Lukáš Hagara
Moduly systému: Modul LeapAR	Martin Mokrý
Moduly systému: Modul Kinect 2	Peter Belai

2. Globálne ciele

V tejto kapitole sa nachádza zoznam globálnych cieľov pre zimný a letný semester. V predmete tímový projekt rozširujeme už existujúci projekt, ktorý disponuje s viacerými funkcionalitami rôznych zariadení.

2.1. Globálne ciele pre ZS

1. **Refaktorizácia zdrojového kódu**
   V projekte sa nachádzajú zoskupenia tried, ktoré by bolo vhodné refaktorovať. Modul, ktoré plánujeme refaktorovať: LuaGraph (dôvod: nedostatočné zapuzdrenie). V rámci refaktorizácie odstránime warningy z kompilátora, cppcheck a cpplint.
2. **Zobraziť model rúk v scéne grafu**
   Pomocou senzoru Leap chceme zobraziť pohyby a gestá používateľov v scéne grafu v reálnom čase. V projekte sa nachádza čiastočná implementácia leap senzoru, ktorú je potrebné dokončiť.
3. **Dataset zo zariadenia Kinect 2**
   Doplniť podporu zariadenia Kinect 2. Kinect 2 bude použitý na získanie datasetu, ktorý budú môcť využívať aj ostatní študenti fakulty. Zabezpečiť, aby si aplikácia vybrala medzi ovládačmi Kinect 1 alebo Kinect 2 podľa pripojeného zariadenia.
4. **Analýza marker less knižníc**
   Nahradenie Aruco knižnicou, ktorá vie pracovať bez markeru. Podmienky na knižnicu sú: jazyk C++, open source, multiplatform, postavená na OpenCV/PCL.

2.2. Globálne cieľe pre LS

1. **Ovládanie grafu pomocou leap senzoru v AR**

   Ako používateľ chcem byť schopný pomocou leap senzoru, web kamery a svojich rúk ovládať graf. Ovládanie má byť možné v obohatenej realite.

2. **Podpora Kinect v2**

   Ako používateľ chcem získat hĺbkovú informáciu obrazu.

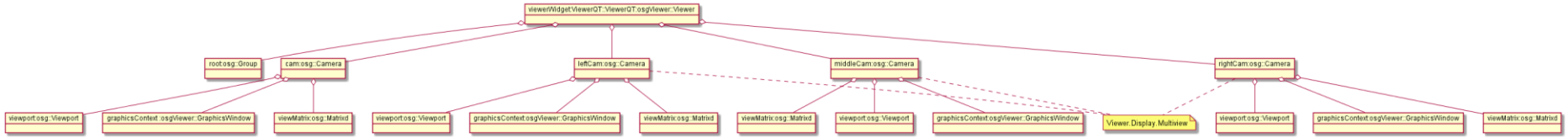
3. **Markerless trackovanie**

   Ako používateľ chcem ovládať systém pomocou gulí, pre jednoduchšie manipulovanie s uzlami grafu. Ovládanie je prevažne používané v argumentovanej realite.

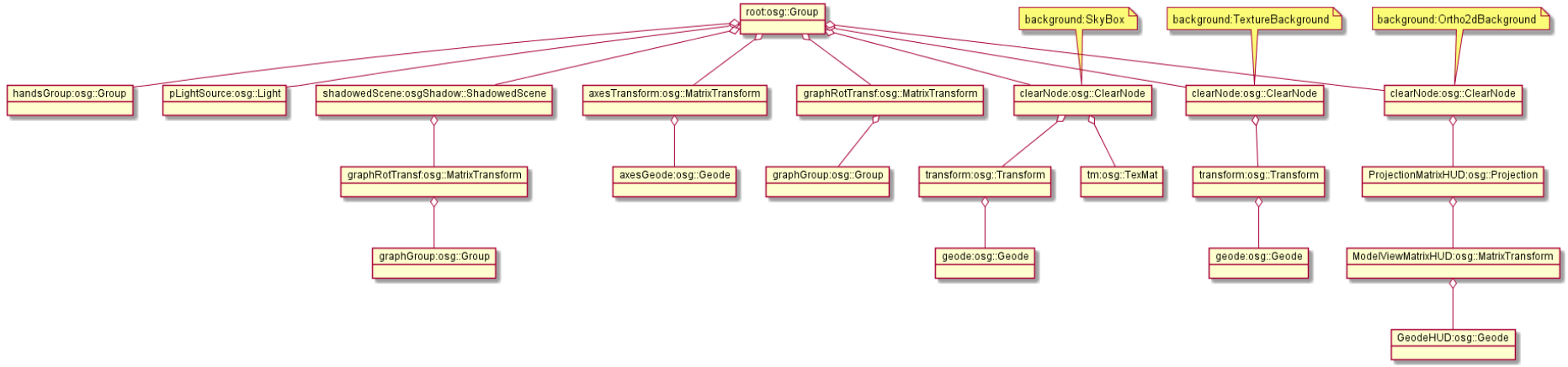
4. **Úprava nití**
   Rozdeliť nite na master niť (celkova logika, spracovanie udajov z device nití) a device(niť získavá údaje zo zariadení a preposiela ich master niti)

3. Celkový pohľad na systém

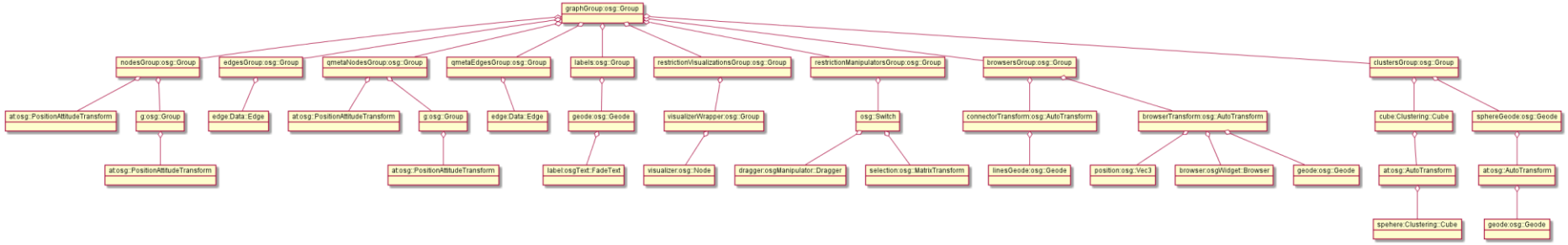
3.1. 0. úroveň abstrakcie(kamery a root)



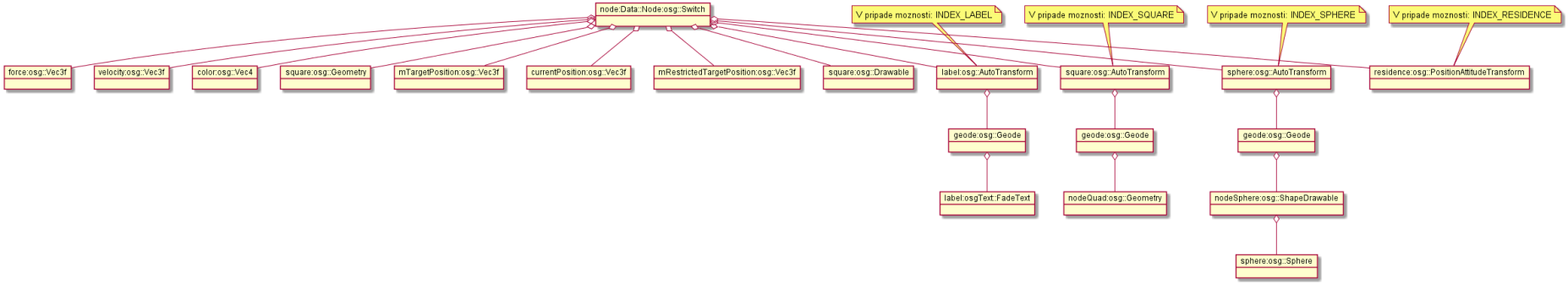
3.2. 1. úroveň abstrakcie(obsah rootu)



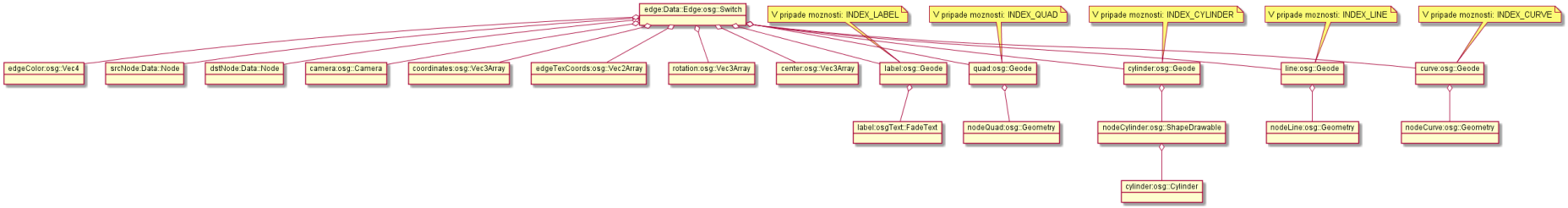
3.3. 2. úroveň abstrakcie(obsah graphGroup)



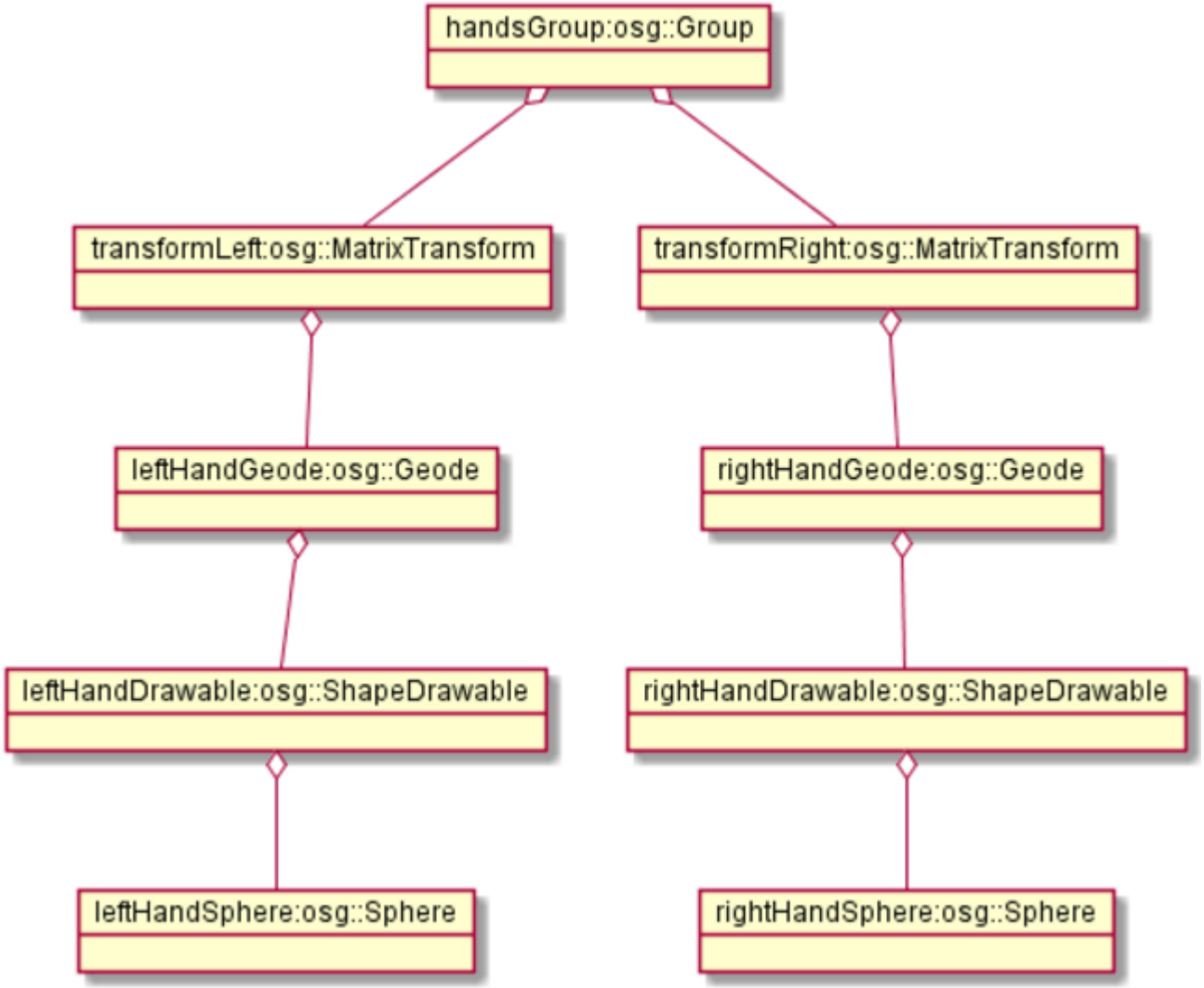
3.4. 3. úroveň abstrakcie(obsah graphGroup)



3.5. Edge(Data::Edge:osg::Switch):

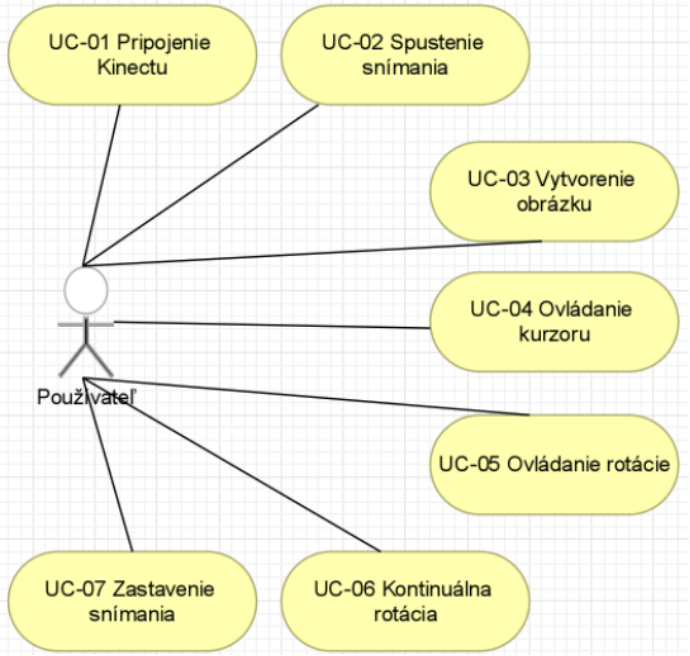


3.6. handsGroup(osg::Group)



3.7. Kinect 2

3.7.1. Use case Kinect 2



Obrázok 3.7.1 Kinect 2 use case

UC-01 Pripojenie Kinectu

1. Používateľ pripojí Kinect do elektrickej siete
2. Používateľ pripojí Kinect do USB
3. Používateľ spustí aplikáciu
4. Systém zobrazí potvrdenie o inicializácii Kinectu

UC-02 Spustenie snímania

1. Používateľ zvolí možnosť zapnutia Kinectu
2. Systém zobrazí okno pre Aruco a Kinect
3. Používateľ zvolí možnosť zapnutia Kinectu
4. Systém zobrazí záznam z live feedu

UC-03 Vytvorenie obrázku

1. Používateľ zvolí možnosť vytvorenia obrázku
2. Systém vytvorí obrázok pomocou normálnej kamery
3. Systém vytvorí obrázok pomocou hĺbkovej kamery
4. Systém vytvorí súbor s hĺbkovou informáciou
5. Systém uloží vytvorené súbory na pevný disk

UC-04 Ovládanie kurzoru

1. Používateľ sa umiestni pred Kinect
2. Používateľ zdvihne ruku s otvorenou dlaňou oproti kamere

3. Používateľ pomocou dlane imituje gesto potlačenia
4. Systém rozpozná dlaň a presunie kurzor podľa jej pozície

UC-05 Ovládanie rotácie

1. Používateľ zvolí možnosť na vypnutie kurzora
2. Používateľ sa umiestni pred Kinect
3. Používateľ zdvihne ruku s otvorenou dlaňou oproti kamere
4. Používateľ pomocou dlane imituje gesto potlačenia
5. Systém rozpozná dlaň a nastaví rotáciu podľa jej pozície

UC-06 Kontinuálna rotácia

1. Používateľ umiestni dlaň na okraj záberu kamery
2. Systém vykoná kontinuálnu rotáciu podľa zvoleného okraju
3. Používateľ umiestni dlaň od okraju kamery
4. Systém zastaví kontinuálnu rotáciu

UC-07 Zastavenie snímania

1. Používateľ zvolí možnosť zastavenie Kinectu
2. Systém zastaví snímanie

4. Funkcie systému

4.1. Ovládanie grafu pomocou leap senzoru v AR

Ako používateľ chcem byť schopný pomocou leap senzoru, web kamery a svojich rúk ovládať graf. Ovládanie má byť možné v obohatenej realite.

4.1.1. Zobrazenie modelu rúk v scéne

Modul LeapAR slúži na zobrazovanie modelov rúk v obohatenej realite. Modely rúk simulujú pohyby vykonávané rukami používateľov v reálnom čase na granularite jednotlivých článkov prstov a dlane. Na svoje fungovanie využíva Leap senzor a jeho podpornú knižnicu.

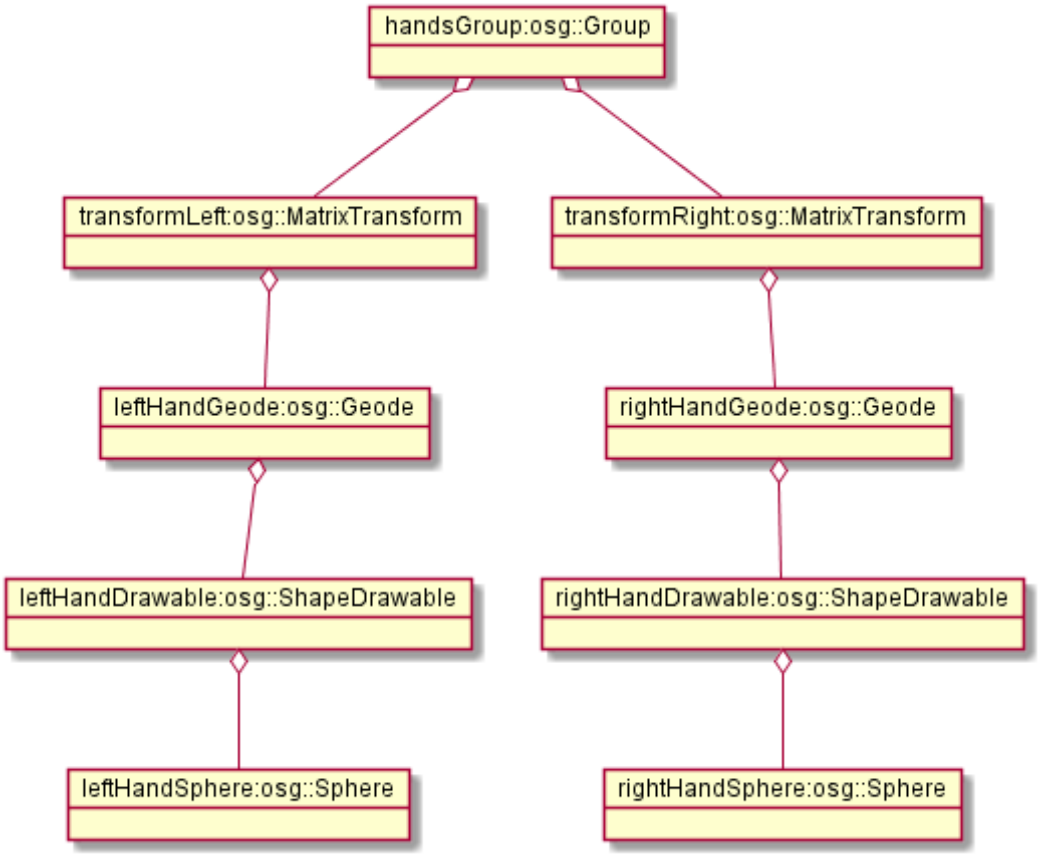
Analýza

V čase vykonávania analýzy existovala základná implementácia tohto modulu, ktorá zachytávala pozície dlaní rúk a prenášala ich do scény vo forme gúľ (angl. sphere), ktoré sa v scéne pohybujú v závislosti od zachytávanej pozície ruky. Pred vykonaním prvotného návrhu bolo potrebné zanalyzovať:

1. Súčasnú štruktúru rúk v scéne
2. Súčasný model adaptéru pracujúceho s Leap knižnicou
3. Triedy a metódy poskytované Leap knižnicou

1. Súčasná štruktúru rúk v scéne:

Každá ruka pozostáva z Transformačnej matice, ktorá obsahuje geódu v tvare gule. Pozícia gule v scéne je menená na základe pozície dlane reálnej ruky.



Obrázok 4.1.1 Súčasný stav štruktúry rúk v scéne

2. Súčastný model adaptéru pracujúceho s Leap knižnicou

Diagram tried adaptéru zobrazuje Obrázok 4.1.2. Popis jednotlivých tried:

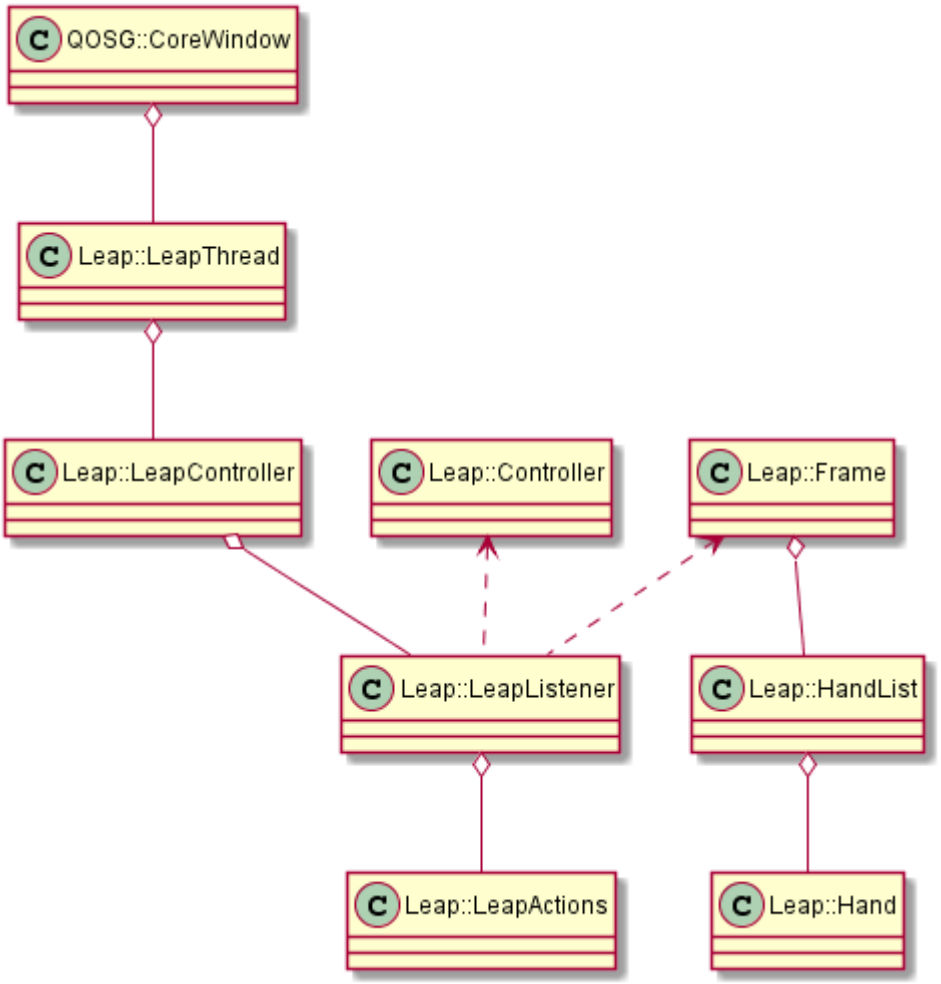
Leap::LeapThread - predstavuje samostatné vlákno, na ktorom sa získavajú dáta z Leap senzoru.

Leap::LeapController - riadi spúšťanie a vypínanie načítavania dát zo senzoru na vlákne.

Leap::LeapListener - dedí od triedy Listener z knižnice Leapu - ma dôležitú metódu onFrame(), ktorá sa vykonáva ak aplikácia beží v popredí.

Leap::LeapAction – získava štruktúrované dáta z Leap knižnice a upravené ich zasiela do objektu na aplikovanie transformácie.

Leap::Frame – trieda knižnice Leapu, ktorá v sebe zaobaľuje Leap::HandList a ten následne Leap::Hand.



Obrázok 4.1.2 Súčasný stav štruktúry adaptéru pre Leap senzor.

3. Triedy a metódy poskytované Leap knižnicou

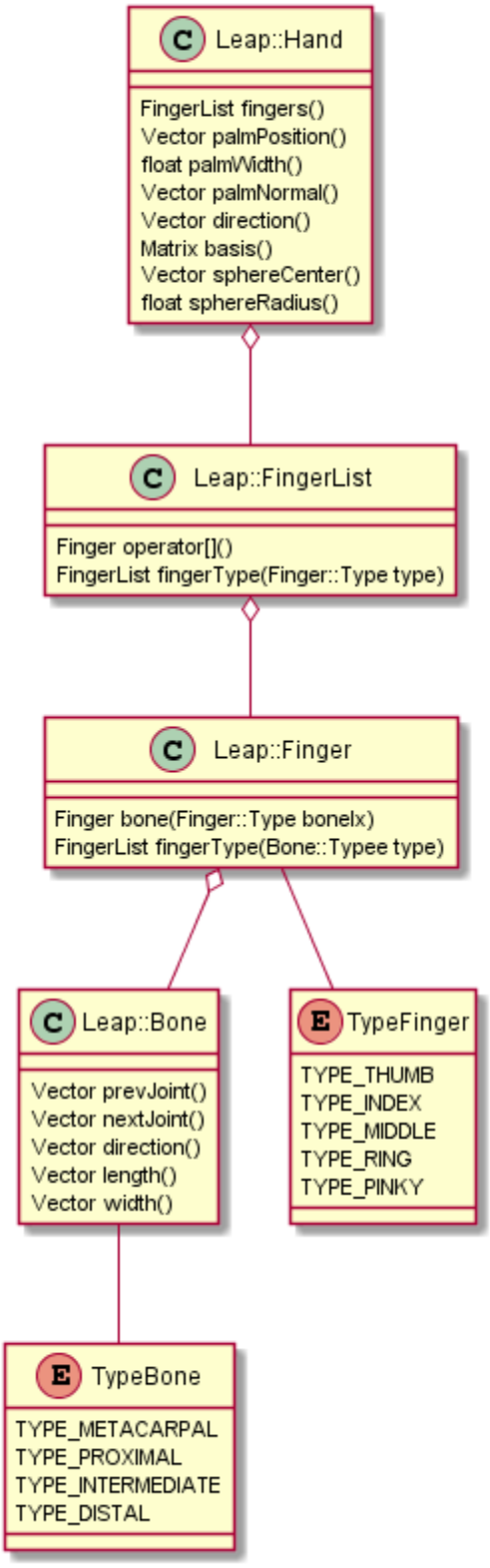
Nižšie uvedený diagram predstavuje zjednodušený pohľad na triedy poskytované Leap knižnicou, pričom zobrazuje triedy a metódy, ktoré sú priamo spojené s pozíciou jednotlivých článkov ruky v scéne.

Leap::Hand – reprezentuje dlaň ruky, z ktorej sa dá získať jej pozícia, natočenie, šírka prstov , ale aj zoznam prstov.

Leap::FingerList – reprezentuje zoznam kostí jednotlivých prstov.

Leap::Finger – reprezentuje samotný prst, ktorý sa skladá z kostí (Leap::Bone) a udržiava si ku každému prstu jeho typ (palec, ukazovák, stredný prst, prstenník, malíček).

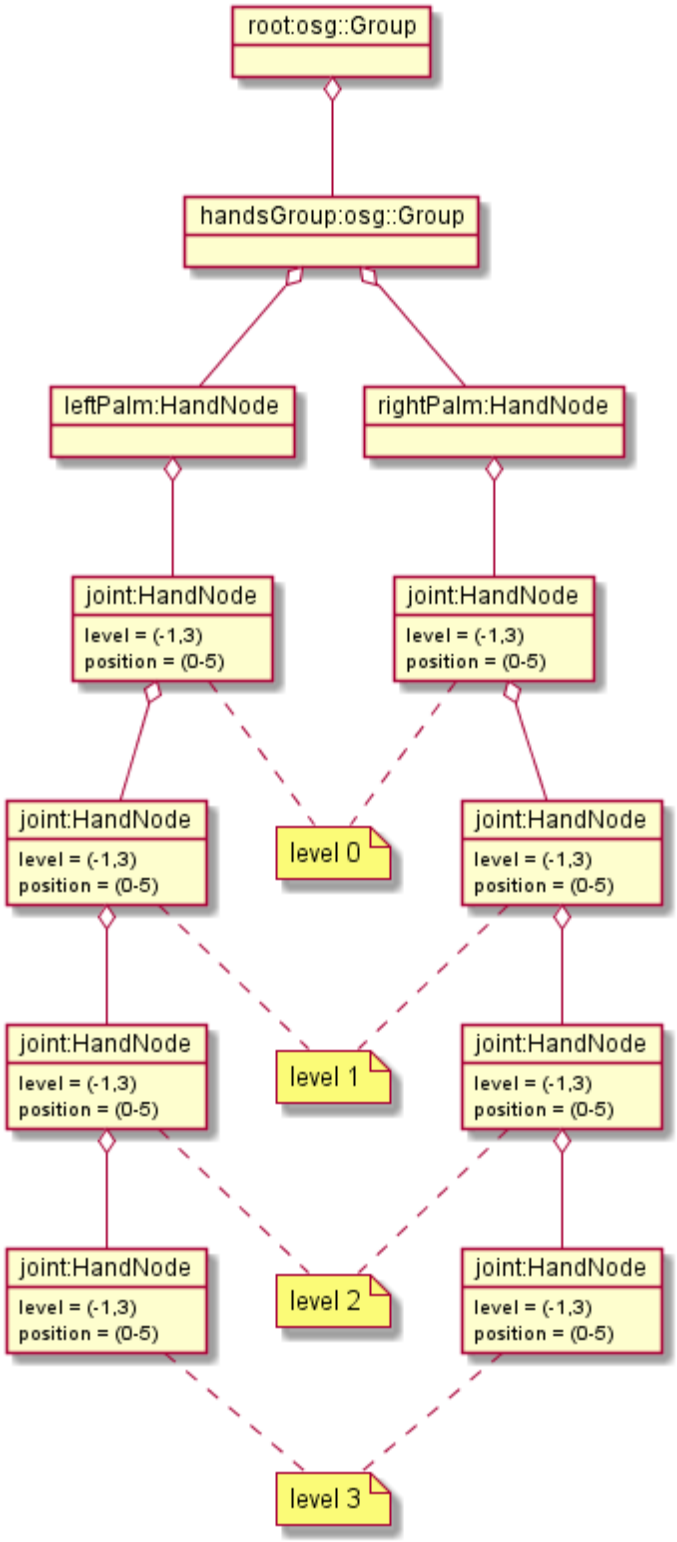
Leap::Bone – reprezentuje kosti nachádzajúce sa v prstoch. Udržiava si informácie o svojom strede, orientácii, dĺžke, šírke, ale aj oboch kĺboch na ktoré je napojená a type kosti.



Obrázok 4.1.3 Dôležité triedy a metódy poskytované Leap knižnicou

Návrh

Prvotný návrh modelu rúk je znázornený v Obrázok 4.1.4. Z pôvodného modulu sa nezmenil iba zaobaľujúci objekt handsGroup. Na rozdiel od knižnice Leapu nebude model ruka vyskladaný z kostí, ale z jednotlivých kĺbov, pričom kosti budú reprezentované ako hrany (spoje) medzi týmito vrcholmi.



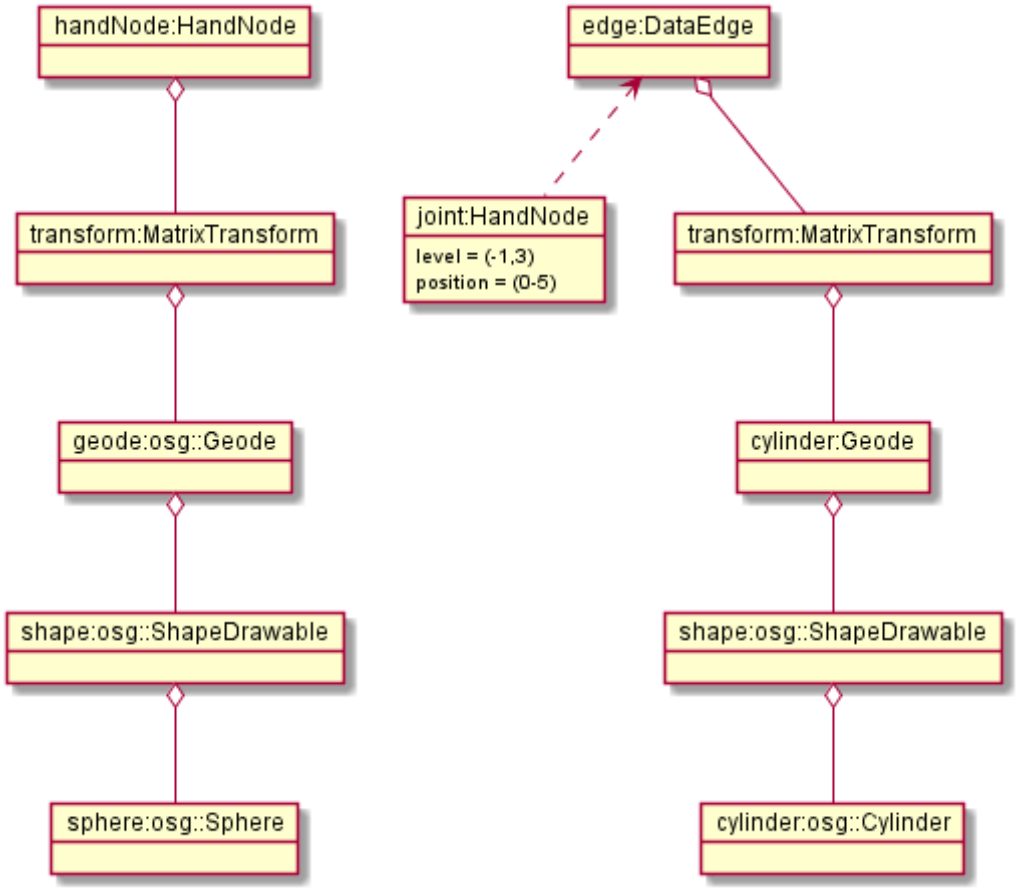
Obrázok 4.1.4. Diagram objektov navrhujúci štruktúru objektov tvoriacu model ruky.

Dlaň reprezentujúca stred ruky, bude znázorňovaná ako kĺb, s tým, že bude mať 5 potomkov typu HandNode, ktoré budú predstavovať začiatky jednotlivých prstov. Ďalší potomkovia týchto kĺbov budú kĺby jednotlivých článkov prstov. Každý HandNode (kĺb) si v sebe bude udržiavať informáciu o svojej pozícii v ruke pomocou atribútov:

- Level – poradie kĺbu v konkrétnom prste.
- Position – typ prstu, ku Ktorému kĺb patrí.

Kĺby budú vizuálne reprezentované pomocou gúľ (osg::Sphere) a kosti pomocou valcov (osg::Cylinder). Vizuálna reprezentácia týchto elementov bude v oboch prípadoch zaobalená do vlastných nadtried (viď Obrázok 4.1.5):

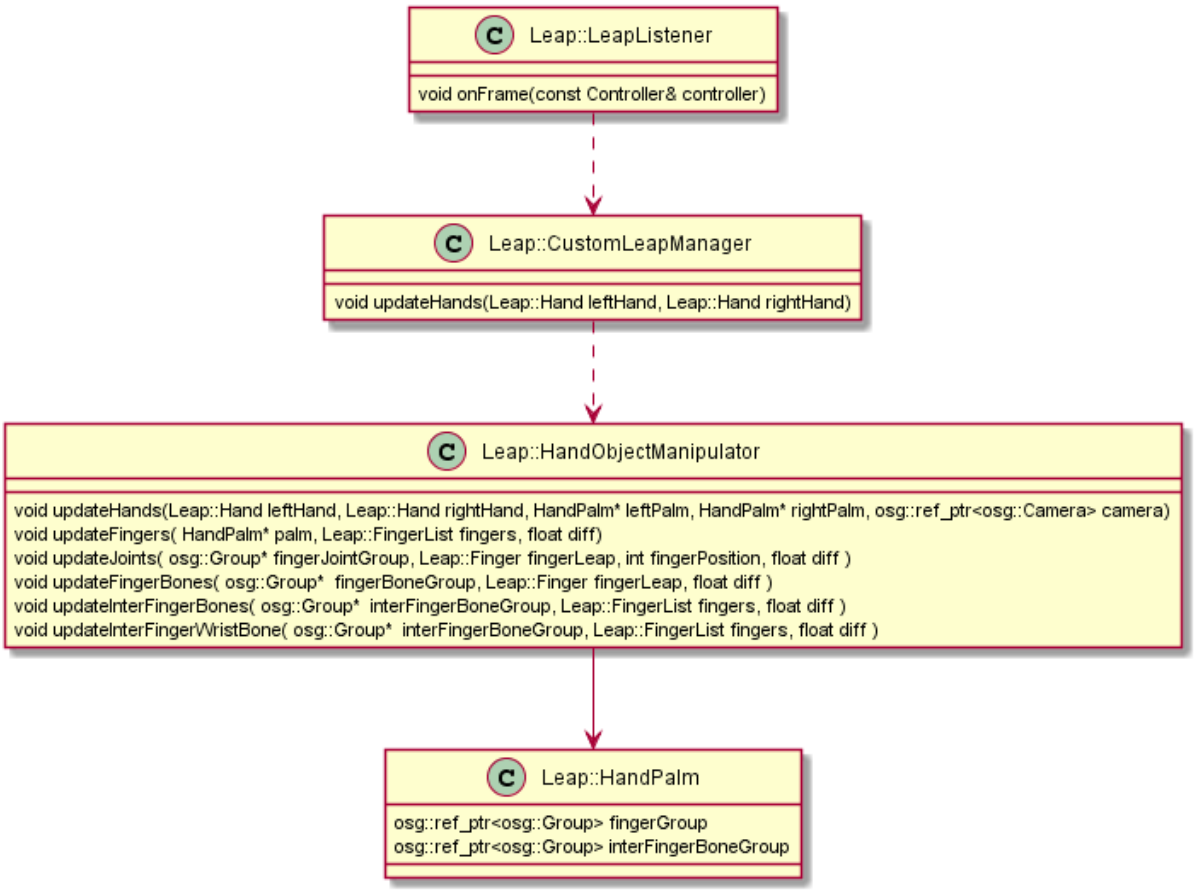
- Kĺb = HandNode
- Kosť = DataEdge



Obrázok 4.1.5. Diagram objektov navrhujúci zakomponovanie vlastných objektov do scény grafu.

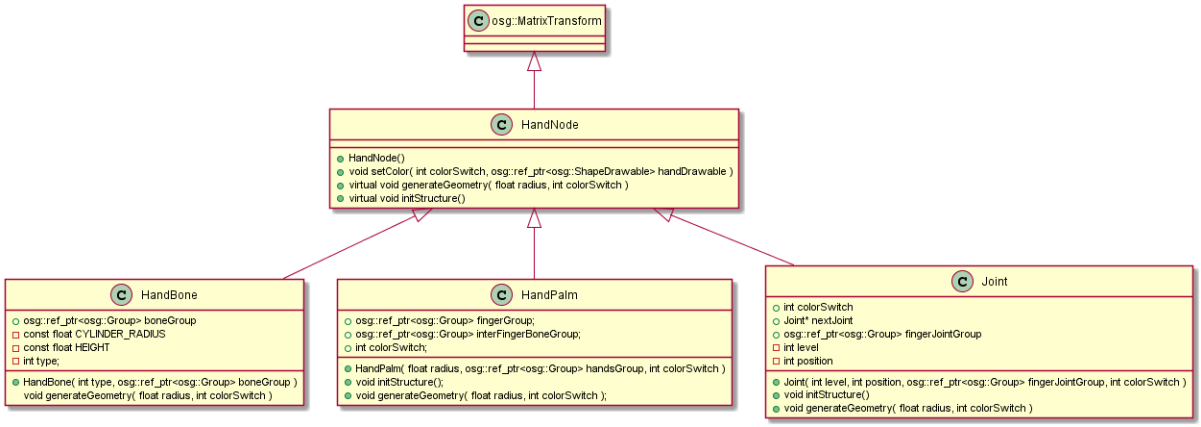
Implementácia

Voči pôvodnému stavu pribudla trieda *HandObjectManipulator*, ktorej účelom je aktualizovať polohu rúk v scéne. Aktualizácia predstavuje komplexný proces úprav polohy a rotácie jednotlivých elementov, z ktorých je model ruky zložený (Obrázok 4.1.7). Triedy podieľajúce sa na procese aktualizácie zobrazuje Obrázok 4.1.6.



Obrázok 4.1.6. Diagram tried zobrazujúci triedy spolupracujúce pri aktualizácii pozície modelu rúk v scéne.

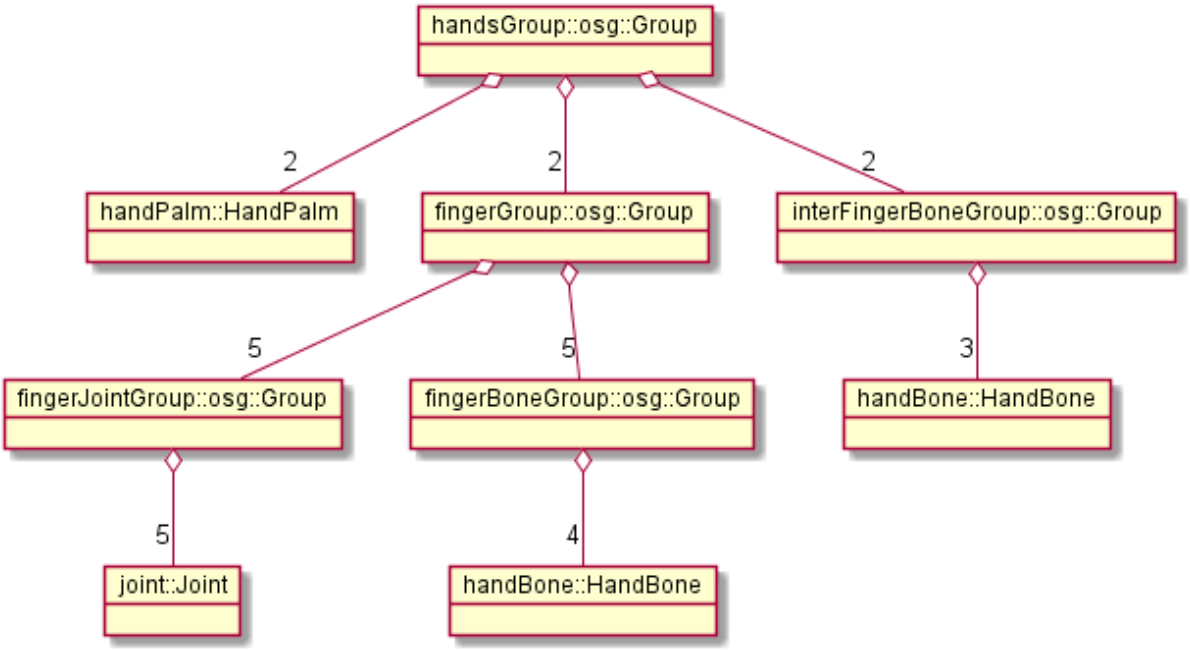
`Privolaním` updateHands sa automaticky postupne spustia aj ostatné a postupne preiterujú celú štruktúru objektov ruky v scéne. Model rúk je zložený z dlane(handPalm), kĺbov (Joint) a kostí (HandBone) viď Obrázok 4.1.7.



Obrázok 4.1.7. Diagram tried zobrazujúci štruktúru tired modelu ruky.

V scéne sú ruky reprezentované osg::Grupou – handsGroup. HandsGroup má v sebe pre každú ruku uloženú dlaň(handPalm), prsty(fingerGroup), medziprtové kosti (interFingerBoneGroup). Prsty osahujú osg:: Group pre každý jeden prst, s tým že prst je zložený z kĺbov a kostí. Medziprtové kosti

sú kosti medzi kĺbami v hánkach a kosť zápästia, ktoré sa v reálnej ruke nenachádzajú, ale dodávajú celistvejší výzor modelu ruky.



Obrázok 4.1.8. Diagram objektov zobrazujúci štruktúru objektov v scéne.

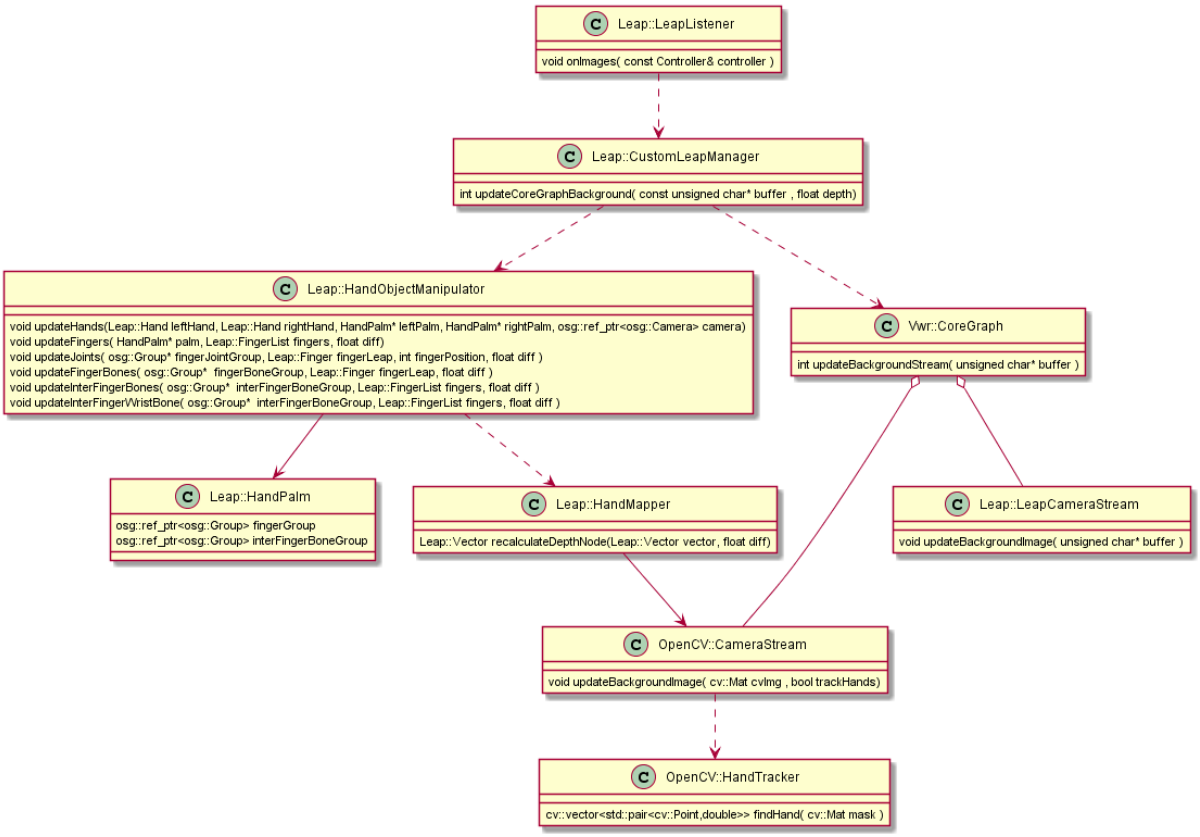
4.1.2. Mapovanie rúk na obraz z kamery

Aby bolo ovládanie pomocou rúk v rozšírenej realite pohodlné je vhodné namapovať model ruky zo scény na ruky zobrazené na kamere. Dosiahnutím tohto cieľu sa model ruky nachádza tam, kde by ho používateľ očakával a zároveň je umožnené identifikovať dotyky modelu rúk s ostatnými objektami v scéne.

Mapovanie bolo implementované pre dva druhy zdrojov obrazu:

- leap senzor
- webkamera

Diagram tried starajúcich sa o mapovanie rúk na obraz pozadia zobrazuje Obrázok 4.1.9. Obraz z Leapu je reprezentovaný triedou *LeapCameraStream*, pričom mapovanie zabezpečuje *HandMapper* prostredníctvom jednoduchej lineárnej transpozície. Jednotlivé obrázky z kamery (angl. frame) zachytáva *LeapListener* prostredníctvom API metódy *onImages*(). Obraz z webkamery spracováva trieda *CameraStream,* ktorá využíva *HandTracker* postavený na knižnici OpenCV na identifikovanie stredu dlane a priemernej vzdialenosti k prstom. Trieda *HandMapper* nastaví odsadenie po x-ovej osi modelu ruky. Odsadenie je vypočítané na základe nameraného rozdielu pozície na monitore a tej, ktorú vráti Leap senzor. Výpočet odsadenia prebieha počas kalibračnej fázy.



Obrázok 4.1.9. Diagram tried zobrazujúce triedy, ktoré majú na starosti spracovanie obrazu z Leapu a webkamery a mapovanie modelu rúk na obraz pozadia.

Návod na použitie

1) Aby mapovanie fungovalo je potrebné, aby bol do scény načítaný graf. napr.
File -> Load graph from file -> circle.graphml.

2) Change Background -> Ortho2D.

3) V ľavej lište:
More Features -> Start LeapAR.

4) V ľavej lište:
More Features -> Start Camera.

5) V menšom okne je potrebné označiť:
Tracking, NoVideo, Background.

V prípade, že je leap smeruje smerom na osobu a nie smerom od nej preč, tak treba označiť aj *Marker is behind.*

6) Presuň menšie okno na bok (obraz sa zobrazuje v hlavnom okne).

7) krok kalibrácie:
Daj ľavú ruku pred kameru, tak aby ju zaznamenával aj leap aj webkamera a drž ju v nehybnej polohe až kým neskončí kalibrácia. Kalibráciu je možné rozpoznať tak, že sa trackuje ruka v scéne a teda sú do obrazu vykresľované obrysy ruky a kružnice vyznačujúce dlaň a prsty.

Pozn. V prípade nepresnej kalibrácie je možné kalibráciu spustiť znova stlačením tlačidla *Recalibrate hand mapping.*

4.1.3. Gestá

Na ovládanie pomocou bolo potrebné implementovať umožňujúce interakciu s grafom. Identifikovali sme nasledujúce gestá:

- Posunutie vystretej dlane - rotácia grafu po osiach x/z/y v závislosti od smeru pohybu dlane
- Dve päste približujúce/vzďaľujúce sa k sebe/od seba - zmenšenie grafu / zväčšenie grafu
- Vystretý ukazovák pretínajúci uzol v grafe - označenie neoznačeného uzlu v grafe / odznačenie označeného uzlu v grafe
- Vykonanie pohybu ukazovákom v smere hodinových ručičiek - otvorenie detailov o označených uzloch
- Vykonanie pohybu ukazovákom proti smeru hodinových ručičiek - zatvorenie detailov o označených uzloch

LeapListener.cpp

Spracováva frame-y pomocou funkcie onFrame, ktoré sú zachytené leapom a zavolá funkciu handleGestures z triedy LeapGestureHandler.cpp.

LeapGestureHandler.cpp

Jeho hlavným cieľom je identifikovanie jednotlivých giest. Každé identifikované gesto následne volá jemu priradenú metódu z agregovanej triedy LeapActions.cpp. V prípade identifikovania gesta na priblíženie / vzdialenie grafu sú ostatné gesta na danom frame ignorované.

Posunutie vystretej dlane

Je zaznamenané gesto swipe, ktoré môže byť vykonané do všetkých možných strán a v závislosti od toho je zavolaná metóda rotateGraph s parametrom Gesture.

Dve päste približujúce/vzďaľujúce sa k sebe/od seba

V prípade, že sú zaznamenané v jednom fram-e dve zovreté päste tak sa skontroluje ich rýchlosť , ktorá je počítaná ako rozdiel ich vektorov pohybu. Ak výsledná rýchlosť presiahla hranicu tak sa graf začne zväčšovať / zmenšovať v závislosti na tom či je hodnota rozdielu vektorov menšia alebo väčšia ako nula.

Vystretý ukazovák

Ak je vystretý iba ukazovák na ktorejkoľvek z rúk a zároveň je najvrchnejší článok ukazováku v stave pretnutia s akýmkoľvek uzlom grafu, zmení pretnutý uzol svoj stav z označeného na neoznačený, respektíve z neoznačeného na označený.

Vykonanie pohybu ukazovákom v smere hodinových ručičiek

V prípade, že sa zaznamená pohyb ukazovákom v smere hodinových ručičiek tak sa zobrazia všetky detaily označených uzlov v načítanom grafe.

Vykonanie pohybu ukazovákom proti smeru hodinových ručičiek

V prípade, že sa zaznamená pohyb ukazovákom proti smeru hodinových ručičiek tak sa skryjú všetky detaily označených uzlov v načítanom grafe.

LeapActions.cpp

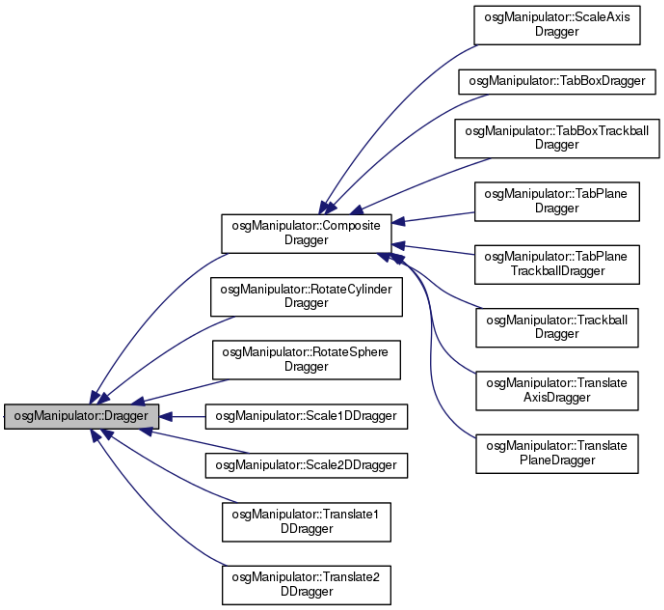
Hlavnou úlohou tejto triedy je obsahovať metódy, ktoré slúžia na vykonávanie logiky za implementovanými gestami z triedy LeapGestureHandler.cpp.

4.2. Manipulátor

Bola taktiež pridaná funkcionalita umožňujúca manipuláciu s grafom za využitia konkrétnych implementácii triedy osg::Manipulator. V súčasnom stave nášho riešenia je možné použiť tieto dva konkrétne manipulátory:

- osgManipulator::TrackballDragger - umožňuje rotáciu objektu vo všetkých troch osiach za pomoci jeho transformačnej matice
- osgManipulator::TabBoxDragger - umožňuje medzi iným aj pohyb objektu vo všetkých troch romzeroch a taktiež umožňuje škálovanie objektu vo všetkých troch dimenziach, avšak nie symetrickým spôsobom

Uvedené manipulátory sa ovládajú pomocou checkbox-ov v triede CoreGraph.cpp. V prípade nutnosti pridania nových manipulátorov je tak možné učiniť v triede CoreGraph.cpp.



Hierarchia dedenia od triedy osgManipulator::Dragger zobrazujúca všetky typy konkrétnej implementácie manipulátorov okrem triedy osgManipulator::CompositeDragger, ktorá je abstraktná.

4.3. Modul Kinect

4.3.1. Analýza

Projekt bol doteraz uspôsobený na zariadenie kinect v1. Avšak nepodporuje v súčasnosti zariadenie kinect v2, ktoré máme aj na fakulte. Z toho dôvodu bolo potrebné zistiť možnosti, pomocou ktorých by bol náš projekt schopný fungovať aj s použitím kinect-u v2.

Na to sú potrebné tieto časti:

- libfreenect2
- openni2
- NiTe 2.2

4.3.2. Návrh

V súčasnosti sa ako najpoužívanejšie a najstabilnejšie riešenie javí opensource projekt na github: https://github.com/OpenKinect/libfreenect2

Projekt je stále vo vývoji, avšak už v súčasnosti funguje pod MacOs X, Linuxom a aj Windowsom. Taktiež sa v projekte nachádza dokumentácia, ako tieto libfreenect2 spojiť s openni2 a Nite 2.2. V prípade nášho projektu bude potrebné meniť rozlíšenie kinect-u v2 na 640 x 480, pretože v opačnom prípade by bolo potrebné prerábať celý projekt. V súčasnosti taktiež zostáva otázne, či bude môcť projekt podporovať obe verzie kinectu súčasne.

4.3.3. Implementacia

V aktuálnej verzii projektu sme rozšírili projekt o podporu pre Kinect v2 na zariadeniach používajúcich unixové systémy. Túto podporu sme dosiahli pomocou vyššie spomínanej knižnice libfreenect2. Zároveň sme tiež opravili chyby ktoré sa predtým nachádzali v minulých verziách modulu Kinect.

4.3.4. Testovanie

Ako testovanie sme skúšali všetku funkcionalitu, ktorú mal modul Kinect poskytovať vo verziách pred naším tímovým projektom. Tú to funkcionalitu sme skúšali na zariadeniach Kinect v2, kde na unixových systémoch tento modul spĺňal, ale bohužiaľ sa nám nepodarilo túto funkcionalitu sprístupniť pre systém Windows.

4.4. Markerless Trackovanie

Ako používateľ chcem ovládať systém pomocou gulí, pre jednoduchšie manipulovanie s uzlami grafu. Ovládanie je prevažne používané v argumentovanej realite.

4.4.1. Analýza

V našej analýze sme sa pozerali na rôzne možnosti využitia existujúcich knižníc na trackovanie objektov bez markera. V tejto analýze sme zistili že naším potrebám dostupné knižnice nevyhovujú. Z tohto dôvodu sme sa rozhodli, že vytvoríme náš vlastný modul na trackovanie bez markerov, ktorý sme implementovali pomocou knižnice OpenCV.

4.4.2. Návrh

Tento modul sme navrhli pôvodne tak, že bude priamo nad kamerou vykonávať operácie a to konkrétne pôvodne hľadanie kruhov pomocou funkcie HoughCircles. Tento návrh sa ukázal ako nevhodný a preto sme ho zmenili. V tomto momente máme dve triedy TrackerBall, ktorá obsahuje údaje o jednotlivých guličkách, a MarkerlessTracker, ktorý zaobstaráva logiku trackovania. MarkerlessTracker je napojený na modul, ktorý sa venuje ovládaniu grafu pomocou markerov, a posiela tam údaje o guličkách ktoré našiel v obraze. Tento obraz sa posiela do nášho modulu z modulu na ovládanie grafu pomocou markerov.

4.4.3. Implementácia

Funkcionalitu tohto modulu sme nakoniec implementovali pomocou algoritmu RANSAC. Tento algoritmus funguje v nasledujúcich krokoch:

1. Vytvorenie hranového obrazu z prijatého obrazu
2. Vo viacerých iteráciách sa náhodne zvolia 3 body hranového obrazu
3. Týmito bodmi sa preloží kruh a vypočíta sa nakoľko sa tento kruh nachádza v hranovom obraze
4. Ak výskyt tohto kruhu v hranovom obraze je vyšší ako zvolená hranica tak ho berieme ako novú guličku

Keďže sme chceli ale ešte zlepšiť výpočtovú efektívnosť tohto algoritmu, tak sme sa rozhodli hľadať výskyty guličiek v jej prechádzajúcich pozíciách, a algoritmus RANSAC púšťať len raz za 5 zaslaných obrazov.

4.4.4. Testovanie

Toto riešenie sme testovali hľadaním viacerých guličiek v živom prenose z webkamery počítača. Riešenie je ešte stále mierne náchylné na kontrast medzi pozadím a guličkami a tiež by bolo vhodné ďalej zlepšovať výpočtovú efektívnosť tohto riešenia. Možnosti zlepšenia výpočtovej efektívnosti sú využívanie algoritmu RANSAC len na časti obrazu kde sa predtým nachádzali guličky a hľadanie nových guličiek pomocou algoritmu RANSAC na celom obraze vo väčších intervaloch.

4.5. Práca Nití

Počas práce na tímovom projekte sme zistili, že v tomto projekte sa počas jeho behu vytvára priveľa nití ktoré pristupujú k rôznym zariadeniam a vykonávajú nad nimi logiku. Toto spôsobilo niekoľko problémov kedy dochádzalo k zbytočnej réžii pri komunikácii medzi týmito niťami, alebo preťaženiu zariadenia, lebo k jeho výstupu pristupovalo viacero nití. Keďže sme toto zistili v dobe, kedy sme nemali už dosť času na prerobenie celej infraštruktúry nití v projekte tak táto časť bude slúžiť ako návrh akým by sa mala práca nití v tomto projekte uberať.

V ideálnom prípade by mala infraštruktúra nití obsahovať dve kategórie:

- **Master Thread –** niť obsahujúca celkovú logiku, získava údaje zo zariadení pomocou všetkých zvyšných nití ale logika zdrojového kódu sa vykonáva len v tejto niti.
- **Device Thread –** niť, ktorá je napojená na jedno zo zariadení a posiela údaje z tohto zariadenia priamo do Master Threadu.