Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Tímový projekt

REKONŠTRUKCIA 3D SCÉNY V2

Inžinierske dielo

Vedúci projektu:

Ing. Vanda Benešová, PhD.

Členovia tímu:

Bc. Jakub Ginter (SI)
Bc. Miroslav Haščič (SI)
Bc. Mário Hunka (SI)
Bc. Viktor Košťan (IS)
Bc. Richard Pintér (IS)

Názov tímu: 3DRecon (tím č. 11)
Web: http://team11-16.studenti.fiit.stuba.sk/
Kontakt: teamfiit11@gmail.com
Akademický rok: 2016/2017
Dátum odovzdania: 15. 05. 2017

OBSAH

1. ÚVOD

Tento dokument vznikol ako technická dokumentácia k projektu 3DRecon, ktorý sme dostali ako zadanie na tímový projekt na FIIT STU. Rekonštrukcia 3D scény je v oblasti výskumu veľmi aktuálnou témou, pretože ma pomerne veľké využitie. Aktuálne existuje veľké množstvo komerčných riešení, ktoré vedia spracovávať dáta zo senzorov skenujúcich 3D scénu. Ich hlavným problémom je, že dokážu spracovávať dáta iba z určitých senzorov (značky).

Našim cieľom bolo vytvoriť systém, ktorý výrazne zjednoduší prácu architektov a dizajnérov a skráti čas vytvorenia modelu priestoru. Bežne sa takýto proces vykonáva ručne prekresľovaním v niektorom počítačovom nástroji. My sme tento proces nahradili naskenovaním priestoru hĺbkovou kamerou, spracovaním týchto dát a ich následným exportom do DXF formátu. Ten je potom ľubovoľne upraviteľný vo všetkých bežne používaných CAD nástrojoch.

Počas dvoch semestrov sme sa snažili pokračovať v práci našich predchodcov a priniesť lepšie výsledky. Podarilo sa nám implementovať možnosť vytvárania vlastného datasetu pomocou skenovania priestoru s využitím zariadenia Kinect One. Vylepšili sme použité segmentačné metódy a algoritmy, ktoré tieto dáta spracovávali, pokúsili sme sa zrýchliť výpočty, vylepšili sme náš vizualizér a pridali možnosť interakcie pomocou zariadení od spoločnosti 3DConexxion. Rýchlosť spracovania taktiež zvýšila možnosť exportu čiastočných výsledkov.

Tento dokument ponúka detailný pohľad na náš produkt a na proces jeho vývoja. Nachádzajú sa v ňom naše čiastočné i globálne ciele ale aj architektúra systému a opis fungovania.

2. GLOBÁLNE CIELE

2.1 Globálne ciele PRE ZS

Projekt sme dostali v rozpracovanej forme preto bolo prvotným cieľom celého tímu oboznámiť sa so systémom, s jeho architektúrou, triedami a celkovým spôsobom fungovania. Súčasne s týmto oboznamovaním sme museli študovať princípy segmentácie 3D scény a používané metódy a algoritmy.

Ďalšie definované ciele na semester boli:

- Refaktoring projektu
    - Zmena štruktúry kódu

    - Odstránenie nepotrebných tried a častí kódu
    - Návrh novej architektúry
    - Tvorba nového GUI
- Analýza možných spôsobov selekcie outliers
    - Analýza dostupných metód
    - Konzultácia v rámci tímu
    - Implementácia vybraných metód
- Získanie nových datasetov
    - Získanie senzoru od spolupracujúcej spoločnosti
    - Naskenovanie nových priestorov
    - Rekonštrukcia týchto dát
    - Testovanie na týchto dátach
- Získanie používaných datasetov
    - Nájsť na internete nové datasety, ktoré sme ešte nepoužili
- Ovládanie vizualizácie prostredníctvom 3D myši
    - Interakcia s vizualizérom pohybmi 3D myšou
- Presnejšie výsledky používaných algoritmov
- Ukladanie medzivýsledkov
    - Uloženie medzivýsledkov pre zrýchlenie výpočtov

Ďalšie úlohy sa špecifikujú postupom času na základe výstupov a výsledok predošlých úloh. Výsledky sa pravidelne konzultujú s vedúcou projektu, ktorá nám radí akým ďalším smerom sa máme pohybovať. Nakoľko tento projekt má výskumný charakter nie je možné dopredu definovať veľké množstvo konkrétnych cieľov, pretože smerovanie sa vždy odvíja od predošlých poznatkov.

2.2 Globálne ciele PRE LS

Po skončení zimného semestra sme pokračovali v práci a snažili sa dokončiť rozpracované úlohy a spojiť jednotlivé oddelené riešenia do jedného celku aby sme v letnom semestri mohli začať pracovať na nových. Hlavným cieľom na tento semester bolo pokračovať v zlepšovaní algoritmov na spracovanie datasetu, implementácia skenovania pomocou hĺbkovej kamery, zlepšenie vizualizácie, zrýchlenie výpočtov a iné.

Zoznam identifikovaných cieľov na letný semester:

- Možnosť ukladania medzivýsledkov segmentačných metód
    - Tým sa zrýchli testovanie a prototypovanie
- Implementácia Kinectu
    - Využitie pre vytváranie vlastných datasetov
    - V rámci celkového použitia sa jedná o veľmi podstatnú funkcionalitu
- Ovládanie vizualizácie prostredníctvom 3D myši

    - Interakcia s vizualizérom pohybmi 3D myšou
- Úprava vizualizéra
    - Zjednodušenie interakcie
    - Pridanie možnosti meniť obsah pomocou jedného tlačidla
- Doxygen
    - Komentovanie kódu podľa metodiky tak aby sa dala vygenerovať dokumentácia
- Lokalizácia outlayerov
- Opravenie zalamovania stien
- Paralelizácia výpočtov
- Implementácia Harrisových hranových bodov

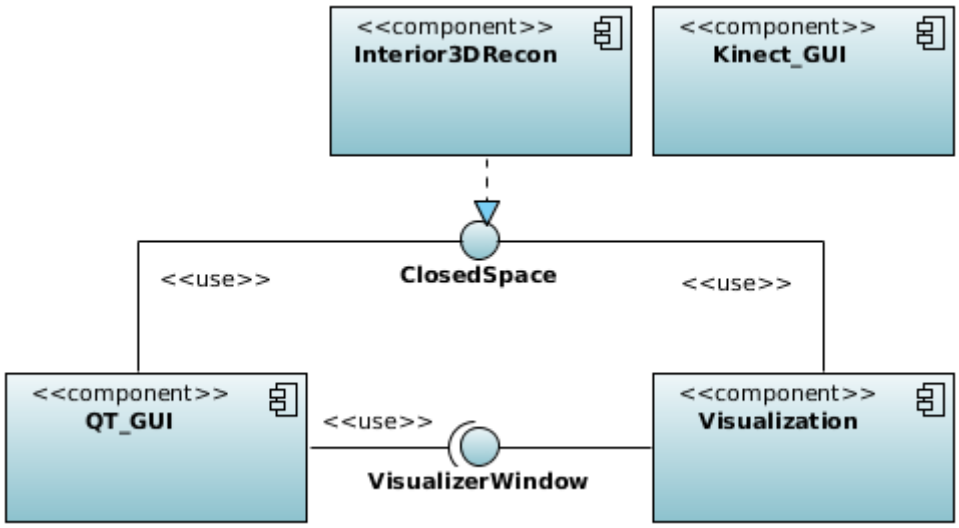
Počas práce na projekte sme často identifikovali nové ciele menšieho významu, na ktorých sme taktiež pracovali. Ciele uvedené vyššie boli však od začiatku našou hlavnou prioritou.

3. CELKOVÝ POHĽAD NA SYSTÉM

V tomto projekte pokračujeme po minuloročnej skupine, ktorá na projekte začínala. Preto sme museli ako prvú vec analyzovať aktuálnu formu projektu. Celý náš tím (5 členov) musel prejsť projekt a pochopiť presné fungovanie jednotlivých tried. Nakoľko bola forma pomerne rozhádzaná a projekt obsahoval množstvo nepotrebného kódu a nepoužívaných funkcií išlo o zdĺhavý proces.

Po tejto analýze sme sa rozhodli, že pristúpime k celkovému refaktoringu tohto projektu a zmeníme jeho štruktúru. Bol navrhnutý nový architektonický model systému založený na predošlom systéme avšak s väčším ohľadom na zameniteľnosť jednotlivých častí a s oddelením používateľského rozhrania od funkcionality. V starom projekte sa časť funkcionality nachádzala aj v projekte používateľského rozhrania čo nám nepripadalo správne v rámci zaužívaných pravidiel a nechceli sme takto postavený projekt ďalej nabaľovať o ďalšiu funkcionalitu.

Nová architektúra je postavená na rovnakých knižniciach ako tá predošlá. Jediný rozdiel je v organizácií projektu. Analýzu možných alternatívnych riešení sme nerobili, pretože sme to nepovažovali za rozumné a časovo efektívne. Aj keby sme našli lepšie riešenia na problematiku segmentácie 3D dát bolo pri by časovo nerentabilné prerábať celý projekt. Preto sme sa zamerali na použitie PCL teda PointCloudLibrary, ktorá poskytuje potrebné rozhranie na prácu s point cloud dátami.. Pridanou častou je projekt pre prácu so skenovacími zariadeniami ktoré využívame na získanie hĺbkových dát. Tieto dáta sú v reálnom čase rekonštruované do spoločného point cloudu a je z nich odstraňovaný šum.



3.1 INTERIOR3DRECON

Interior3DRecon predstavuje hlavnú funkcionalitu nášho projektu. Obsahuje triedy na segmentáciu, import a export 3D dát.

Jej aplikačné rozhranie predstavuje trieda *CosedSpace*, ktorá v sebe zoskupuje načítané pôvodne 3D dáta, konfiguračné parametre a metódy na výpočet/prístup ku segmentovaným dátam. Je navrhnutá tak, aby jej inštancia obsahovala segmentované dáta pre práve jednu kombináciu pôvodných dát a konfiguračných parametrov. Je to preto, aby sa zabezpečila lepšia viditeľnosť a umožňuje to jednoduchšie a prehľadnejšie ukladanie dát inštancie.

Jednotlivú segmentáciu 3D dát vykonávajú podtriedy triedy *Segmentation*. Podobne ako trieda ClosedSpace sú navrhnuté tak, aby si po inicializácii a vykonaní *calculate* metódy uchovali vypočítané dáta a v prípade, že bude potrebné segmentovať iné dáta alebo použiť inú konfiguráciu vytvorí sa nová inštancia. Umožní to uchovať si výsledky tej istej segmentačnej metódy s rôznymi konfiguráciami, čo sa môže hodiť v prípade, že sú potrebné tieto rozličné výsledky na ďalšie spracovanie.

V súčasnom projekte sú implementované tri triedy, ktoré dedia od triedy Segmentation - *BirkysNormalSegmentation*, *AnalyticalPlanesSegmentation* a *PlanePointsExamination*. *BirkysNormalSegmentation* segmentuje point cloud na základe odhadnutých normál bodov. Tieto segmentované dáta berie na vstupe *AnalyticalPlanesSegmentation*, ktorá ďalej segmentuje 3D scénu na planárne polygóny pomocou fitovania rovín cez klastre bodov zo vstupu. *PlanePointsExamination* odstraňuje outliere z klastrov, ktoré definujú vypočítané polygóny. Táto trieda je už zastaralá a odstraňovanie outlierov sa presunulo do triedy *AnalyticalPlanesSegmentation*.

Triedy *CommonUtil* a *DataStatistics* obsahujú podporné metódy pre podtriedy triedy *Segmentation*.

Metódy a funkcie na import a export dát sú obsiahnuté v triedach *Exporter* a *DataReader* pričom *DataReader* používa triedu *InputDataParser*, ktorá parsuje ľubovoľné ASCII dáta do point cloudu v XYZ koordinačnom systéme.

3.2 QT_GUI

Pomocou tohto modulu je spúšťaný celý projekt. Zvyšné moduly sú kompilované do knižníc, ktoré *QT_GUI* používa. Jeho jediná funkcionalita je pritom používateľské okno, ktoré umožňuje volať funkcionalitu modulov *Interior3DRecon* a *Visualization*.

Jeho jediná rovnomenná trieda *QT_GUI* obsahuje premennú typu *ClosedSpace,* ktorá poskytuje aplikačné rozhranie modulu *Interior3DRecon* a premennú typu *Visualizer*, ktorá poskytuje aplikačné rozhranie modulu *Visualization*. Po vybratí súborov s 3D dátami a konfiguračného súboru sa zavolá inicializačná metóda triedy *ClosedSpace* a sprístupnia sa tak tlačidlá, ktoré volajú jej zvyšné metódy a metódy *Vizualizéru.*

Na vytvorenie používateľského rozhrania sme použili populárny softvér *QT*. Umožnil nám pomerne rýchlo vytvoriť grafické používateľské rozhranie.

3.3 VISUALIZATION

Vizualizáciu sme dokázali kompletne oddeliť od používateľského rozhrania a pomocné výpočtové hodnoty posielame z GUI ako parametre, takže v GUI je iba volanie funkcie. Vizualizér pracuje s inštanciou ClosedSpace, ktorá je hlavnou inštanciou celého programu a všetky operácie sa robia nad ňou, resp. všetky vizualizované dáta získavame z nej.

Vizualizácia prebieha s využitím metód knižnice PCL. Metódy vyberáme na základe pointcloudu, ktorý chceme vizualizovať, teda podľa toho aké ma vlastnosti. Jeho vlastnosti sa menia postupným spracovávaním prostredníctvom implementovaných metód. Spôsob vizualizácie sme úplne zmenili. Po vybraní konkrétnej vizualizácie sa najskôr zavolá metóda, ktorá vytvorí prázdne okno vizualizéru a následne sa zavolá iná, ktorá ho naplní podľa požiadaviek. Takto sme sprehľadnili kód a umožnili používateľovi dynamicky za behu programu meniť obsah vizualizéru stlačením jedinej klávesy. Túto funkcionalitu sme implementovali z dôvodu porovnávania datasetov pred a po spracovaní konkrétnou metódou.

Máme vytvorený vlastný spôsob interakcie ktorý sme prebrali od minuloročného tímu ale bolo potrebné prepracovať ho. Zmenili sme takmer všetky metódy interakcie. Hlavné problémy, ktoré sme identifikovali bolo príliš veľké posúvanie sa o konštantnú vzdialenosť bez ohľadu na veľkosť datasetu. Všetko posúvanie sme zmenili tak, aby sa kamera vždy posunula o percentuálnu hodnotu veľkosti datasetu. Iným problémom bolo, že niektoré datasety boli posunuté v priestore, teda sa ich stred nenachádzal v bode [0,0,0] kam sa pozerala kamera. Museli sme implementovať metódu výpočtu stredu datasetu a nastaviť kameru vždy na tento bol v súradnicovej sústave. Taktiež sme pridali množstvo možností rotovania datasetu pomocou numerickej klávesnice. Všetky možnosti interakcie s vizualizérom sú spísané v príručke k vizualizácií.

Ďalšou úpravou interakcie a celkovej vizualizácie je pridanie 3D Myši. Podporu 3D zariadenia od spoločnosti 3dConexxion sme implementovali do nášho vizualizéru a vytvorili sme vlastné interakčné metódy. Pôvodne sme sa snažili využiť prázdne implementácie, ktoré priamo pre toto zariadenie

poskytuje VTK, avšak narazili sme na niekoľko problémov a preto sme sa rozhodli vytvoriť tieto metódy vlastnou cestou. Po niekoľko násobnom testovaní a upravovaní metód sa nám podarilo vytvoriť také, ktoré vyhovovali väčšine používateľom.

4. 3DRECON_KINECT

Tento modul predstavuje nástroj pre prácu s senzormi pre skenovanie vnútorných priestorov. Obsahuje hlavné rozhranie z ktorého sú volané ďalšie časti projektu. To je implementované pomocou softvéru *QT* podobne ako modul QT_GUI.

Pre prácu s implementáciou ktorá sa nachádza v tom istom projekte obsahuje triedy *kinfuRunner* ktorý predstavuje implementáciu SLAM algoritmu pomocou PCL knižnice s vlastnými úpravami. Ďalej obsahuje menej náročnú implementáciu kinectDriverTest ktorej úlohou je iba poskytnúť nástroj na otestovanie pripojenia senzoru.

5. MODULY SYSTÉMU

Väčšinu modulov systému sme prevzali po predošlom tíme. Tie sú podrobne zdokumentované na ich prezentačnej stránke[1].Vytvorili sme tiež tri nové moduly *Kinect*, *CustomInteractor* a *Serialization*:

- **BirkysNormalSegmenation** - segmentácia point cloud do klastrov na základe vypočítaných normál bodov.
- **AnalyticalPlaneSegmentation** - (*pôvodne AnalyticalPlaneReconstruction*) segmentácia planárnych polygónov z jednotlivých klastrov vypočítaných pomocou *BirkysNormalSegmentation*.
- **CustomInteractor** - ovládanie PCL vizualizačného okna.
- **Načítavanie konfiguračných údajov z XML** - načítavanie konfiguračných údajov (opensource implementácia TinyXML).
- **Exporter** - export dát do formátov *pcd* a *dxf*.
- **Kinect -** skenovanie priestorov pomocou senzorov a rekonštrukcia 3D scény.
- **Serialization** - serializácia segmentačných tried, kvôli ukladaniu a načítaniu medzivýsledkov.

[1] Prezentačná stránka tímu *RED*

link: http://labss2.fiit.stuba.sk/TeamProject/2015/team05is-si/#final-product

5.1 ROZPOZNÁVANIE A SEGMENTÁCIA STIEN

5.1.1 ANALÝZA

Na základe toho, že kvalita výsledkov segmentácie a jej výpočtová náročnosť nebola postačujúca na ďalšie spracovanie, bola implementácia analyzovaná vzhľadom na iné práce, ktoré sa venovali rozpoznávaniu a segmentácii stien z point cloudu.

5.1.2 METÓDA ROZPOZNÁVANIA STIEN

Rozlišujeme 3 skupiny metód na rozpoznávanie rovín v point cloude:

- Region growing techniky
- Metódy založené na RANSAC-u
- Metódy založené na transformácií priestoru príznakov (napr. Houghova transfornácia)

Súčasná implementácia používa PCL implementáciu RANSAC-u. Práca [1] navrhuje použitie metódu rozpoznávania stien pomocou Houghovej transformácie, ktorá dosahuje stabilnejšie výsledky pri rozpoznávaní stien zo sekvencie 3d dát v čase pričom si zachováva porovnateľnú výpočtovú náročnosť. Keďže ale náš projekt rieši rozpoznávanie zo statických 3d dát je táto výhoda nepodstatná.

RANSAC má nevýhodu, že bez optimalizácie je výpočtovo náročný. Navzdory tomu bolo ukázané, že dokáže sa vysporiadať s datasetom obsahujúcim 50% outlayerov. Práca [3] rieši rozpoznávanie tvarov z point-cloudu použitím optimalizovaného RANSAC algoritmu pomocou hodnotiacej funkcie. V jednotlivých iteráciách tohto algoritmu sa vyberie tvar s maximálnym ohodnotením zo skupiny kandidátov, zistí sa pravdepodobnosť, že nebol prehliadnutý lepší kandidát a v prípade, že je táto pravdepodobnosť dostatočne vysoká vybraný kandidát je vymazaný zo skupiny kandidátov. Algoritmus končí v prípade, ak pravdepodobnosť, že v skupine kandidátov nie je lepší kandidát ako používateľom definovaný minimálny tvar je dostatočne veľká.

Súčasná implementácia používa na fitting rovín RANSAC nad segmentmi point-cloudu, ktoré boli segmentované pomocou dominantných normál point-cloudu a následného použitia Region growing algoritmu. Je teda primárne zameraná na rozpoznávanie planárnych objektov, pričom si vyžaduje pomerne výpočtovo náročný preprocessing. Preto stojí na zváženie použitie RANSAC algoritmu spolu s optimalizáciou, podobne ako to bolo v prípade práce [3].

5.1.3 VALIDÁCIA ROZPOZNANÝCH ROVÍN

Po fittingu stien pomocou RANSAC-u sa rozpoznané roviny evaulujú, či segment skutočne predstavuje planárny objekt, resp. povrch na základe pomeru 2:1 inlayers a outlayers. V prípade, že outlayer bodov je viac ako polovica inlayer bodov je táto stena (rovina) zamietnutá a ďalej sa s ňou neráta.

V tejto implementácii vidieť nedostatok, pretože sa takto môže zamietnuť segment, ktorý obsahuje stenu alebo planárny povrch s iným priľahlým objektom. Preto sa navrhuje implementovať odstránenie potencionálneho priľahlého objetku napríklad odstránením bodov segmentu, ktoré sa nachádzajú za určitou vzdialenosťou od rozpoznanej roviny.

5.1.4 ZLEPŠENIE PRESNOSTI STIEN

Pre práce, ktoré rozpoznávajú steny pomocou RANSAC-u je obvyklé, že po rozpoznaní steny nasleduje vylepšenie (refining) steny pomocou algoritmu najmenších štvorcov. Optimalizuje sa tak chyba určenia roviny v segmente.

Toto vylepšenie určenia roviny steny chýba v súčasnej implementácii. Je preto na zváženie jeho použitie po odstránení outliers ako posledný krok fitovania rovín, ktorý minimalizuje štandardnú odchýlku.

5.1.5 ZALAMOVANIE STIEN V SÚČASNEJ IMPLEMENTÁCIÍ

Na základe analýzy segmentačného algoritmu predošlého tímu sa zistila príčina zalamovania polygónov. Pri zjemňovaní okrajových bodov konkávnej obálky sa tieto body transformovali na najbližšie body segmentu, ktoré ale neležia na jednej rovine a preto nebol výsledný polygón planárny. Bolo potrebné ešte transformovať tieto body pomocou kolmých priemetov na analyticky určenú rovinu segmentu.

5.1.6 ODKAZY

[1] HULIK, Rostislav, et al. Continuous plane detection in point-cloud data based on 3D Hough Transform. Journal of visual communication and image representation, 2014, 25.1: 86-97.
[2] ROTHG., LEVINE M. D.: Extracting geometric primitives.CVGIP: Image Underst.58 , 1 (1993), 1–22.
[3] SCHNABEL, Ruwen; WAHL, Roland; KLEIN, Reinhard. Efficient RANSAC for point-cloud shape detection. In: Computer graphics forum. Blackwell Publishing Ltd, 2007. p. 214-226.

5.1.7 NÁVRH A IMPLEMENTÁCIA

Na základe analýzy sa navrhlo riešenie zalamovania stien. Riešenie spočíva v premietaní okrajových bodov jednotlivých polygónov (stien) na rovinu daného polygónu pomocou triedy PCL knižnice *ProjectInliers*.

5.1.8 TESTOVANIE

Riešenie zalamovania bolo overené automaticky testom aj manuálne vizuálnym overením. Automatický test spočíval v dosadzovaní súradníc okrajových bodov polygónu do parametrickej rovnice roviny polygónu.

5.2 ÚPLNÁ SEGMENTÁCIA OUTLAYEROV

5.2.1 ANALÝZA

Analýza požiadaviek prebieha počas spoločných stretnutí s produkt ownerom. Na základe diskusií sme identifikovali problém s tzv. *outlayer-mi* - výčnelkami (kľučka, svetlo apod.). Problém úzko súvisí so segmentáciou polygónov, na ktorú je využitá metóda opísaná vyššie (***AnalyticalPlaneReconstruction***). Odstránením tohto problému je možné značne vylepšiť presnosť segmentácie polygónov, ktoré sú kľúčové pri identifikácií stien či objektov.

Konkrétne, segmentácia polygónov prebieha v týchto krokoch. Z troch bodov sa definuje rovina, táto rovina sa preloží cez vytvorené segmenty. Pri evaluácií sa zistí koľko percent zo segmentovaných klastrov sa pretína s definovanou rovinou. Samozrejme, je potrebná určitá odchýlka, ktorá určuje kedy bod patrí a kedy nepatrí do roviny. Odchýlka je určená vzhľadom na šum, ktorý vstupné dáta obsahujú. Náš problém spočíva teda v tom, ako oddeliť šum od reálnych objektov. Napríklad, sme si vedomý toho aký nepresný je náš hardware. Avšak, objekty, ktoré sú relatívne malé vzhľadom na snímanú miestnosť - napr. kľučka - sa nám javia ako šum - čo ale nie je pravda. V prípade, že by sme tieto objekty vedeli identifikovať a nejakým spôsobom ich zo scény odstrániť (vysegmentovať) vieme zaručiť lepšiu identifikáciu polygónov.

Analyzovali sme viacero spôsobom, ktorými je možné lokalizovať a následne segmentovať outlayeri. Jedným z nich je ABOD (Angle-Based outlayer detection). Pri tejto metóde zisťujeme kde sa bod nachádza vzhľadom na ostatné body. Ak sa "pozerá" na väčšinu ostatných bodov v jednom smere - je outlayer. Ak existuje viacero smerov, v ktorých vidí body - outlayer nie je. Pri tejto metóde je hlavná nevýhoda jej zložitosť a to O(n^3). Ďalším spôsob je využitie neurónovej siete. Avšak tu nastáva problém s označkovaným datasetom. ABOD by bolo možné optimalizovať tým, že nám používateľ označí miesta, kde sú potenciálne outlayeri. Tým pádom, by sme nemuseli prehľadávať celý point cloud, ale iba jeho časti. Avšak, naše riešenie má byť plne automatizované, takže táto možnosť nepripadá do úvahy. Nakoniec sme sa rozhodli pre otestovanie metódy, ktorá je opísaná nižšie v Návrhu.

5.2.2 NÁVRH A IMPLEMENTÁCIA

Navrhli sme metódu, ktorá je založená na projekcií z 3D do 2D. Prebieha v týchto krokoch:

1. StatisticalOutlierRemoval - túto funkcionalitu ponúka PCL. Dokáže zabezpečiť redukovanie šumu z 3D dát.
2. Projekcia do 2D - spravíme prierez segmentu, čím sa premietne do 2D.
3. Best line fit - cez body sa snažíme preložiť priamku, ktorá pretne čo najviac bodov
4. Lokalizácia outlayers - body, ktoré ležia v kolmej vzdialenosti +- treshold, sú outlayeri
5. Outlayers sú nakoniec vymazané a do znovu počítania *AnalyticalPlaneSegementation* sa dostávajú iba *Inliers*.

5.2.3 TESTOVANIE

Evaluácia výsledkov riešenia tohto problému bude prebiehať nasledovným spôsobom. Je potrebné vždy poznať aj výsledky starej metódy, preto je potrebné si ich niekam uložiť. Pri tej istej konfigurácií, identifikujeme polygóny bez toho aby sme odstránili outlayer-i a uložíme si výsledky o úspešnosti (koľko percent bodov zo segmentovaných častí patrilo do roviny). Následne pripojíme našu metódu segmentácie outlayerov a znovu spustíme proces identifikácie polygónov. Výsledky vieme teda ľahko pomocou tejto metriky porovnať. Ak dosiahneme značne lepšie výsledky metóda bude integrovaná do projektu.

Je potrebné pôvodný výstup kontrolovať aj ručne - pomocou vizualizéru, a teda porovnať výstup pôvodných metód v podobe point cloud s výstupom nových metód segmentácie a identifikácie outlier-ov.

5.3 REKONŠTRUKCIA DÁT Z KINECTU

5.3.1 ANALÝZA

Získavanie testovacích dát - voľne dostupné datasety na internete, datasety od firmy *Photoneo s.r.o.* - obmedzuje vývojoví tím pri testovaní nových výskumných metódach. Ak by sme chceli dataset šitý na mieru, je časovo náročné (možno aj nemožné) takýto dataset zabezpečiť. Vzhľadom nato, sme na spoločných stretnutiach prišli k záveru, že je vhodné implementovať vlastnú rekonštrukciu dát.

5.3.2 NÁVRH

Preskúmali sme viacero možností ako pri skenovaní miestnosti minimalizovať odchýlky, ktoré môžu spôsobiť zlé mapovanie bodov v priestore. Všetky preskúmané možnosti využívali určitú variantu **SLAM algoritmu (simultaneous localization and mapping).** Algoritmus je rozdelený do dvoch častí: mapovanie a lokalizácia. mapovanie predstavuje proces nadväzovania a odstavovania chýb zo skenovaných dát. Lokalizačná časť je zase zameraná na výpočet polohy a správne rozpoznávanie natočenia senzoru, polohy v priestore atď. Problémom môže byť náročnosť algoritmu na výpočtový výkon a špeciálne prípady pri SLAM algoritme ako je opätovné navštívenie oblastí. Po preskúmaní možností sme sa rozhodli pre použitie implementácie ktorá sa nachádza v knižnici PCL a jej nasledovné upravenie pre naše potreby. nasledujúceho algoritmu:

5.3.3 IMPLEMENTÁCIA

Implementáciu sme sa kvôli náročnosti na výkon rozhodli úplne oddeliť od hlavného projektu rekonštrukcie. Bol vytvorený vlastný solution pre implementáciu SLAM algoritmu. Pre zvýšenie použiteľnosti sme vytvorili používateľské rozhranie pomocou *QT*. Ako prvým krokom bola

implementácia získavania dát zo senzorového zariadenia. Keďže sme mali k dispozícií hlavne senzor Kinect One bolo primárnym cieľom získavať dáta práve z tohto senzoru. Pretože implementácia vlastného ovládača pre zbieranie dát zo zariadenia by bola príliš náročná rozhodli sme sa použiť knižnicu OpenNI2 ktorú sme upravili tak aby použila ovládač získaný od tretej strany. Pomocou tejto knižnice a PCL vizualizéra sme implementovali základný testovací nástroj na overenie spojenia medzi počítačom a senzorom. Tento nástroj zobrazuje hĺbkové dáta zo senzoru v reálnom čase. Ďalej sme sa pokúsili prebrať knižničnú implementáciu podobného nástroja, tá však nepodporovala zbieranie dát z OpenNI2 a tak sme z implementácie použili iba niektoré časti a vytvorili vlastnú. Tá je založená na moduloch knižnice PCL, predovšetkým modul GPU ktorý spravuje využitie grafických kariet nVidia. Tento modul sme museli čiastočne upraviť aby podporoval novšie architektúry grafických kariet keďže definície v PCL podporujú iba architektúry nižšie ako maxwell. Druhým modulom ktorý využívame je implementácia Kinfu. Je to implementácia SLAM algoritmu v PCL knižnici. Tú sme upravili aby pracovala s OpenNI2 ovládačom pre skenery a grafickými kartami všetkých architektúr Pre budúce potreby sme implementovali zbieranie farebných obrazov počas skenovania ktoré je možné namapovať na výsledný PointCloud. Po spustení skenovania sa používateľovi zobrazia tri okná, prvé zobrazuje aktuálne hĺbkové dáta zo senzoru, druhé rekonštruovaný PointCloud a tretie polohu skenera v priestore spolu s indikátorom stratenia polohy.

5.3.4 TESTOVANIE

Testovanie riešenia tohto problému prebiehalo manuálne s použitím vizualizačných okien v rámci skenu a následné overenie pomocou projektu 3DRecoon. Skenovanie sa ukázalo ako presné a rýchle pri dostatočne výkonnom hardvéry. Problémom bolo pomerne časté strácanie polohy senzoru. To sa však dá redukovať pomalým procesom skenovania a plynulými pohybmi so senzorom. Bolo otestované iba zariadenie Kinect One, ale iné zariadenia by nemali predstavovať veľký rozdiel vo výsledkoch, keďže jediným ovplyvňujúcim faktorom je presnosť hĺbkových dát senzoru z ktorých vieme pomerne presne odstraňovať šum. Samozrejme je optimálnejšie použiť presnejší senzor pre dosiahnutie lepšieho skenu.

5.4 ZLEPŠENIE VIZUALIZÁCIE

5.4.1 ANALÝZA

V rámci nášho projektu je vizualizácia veľmi podstatná časť. Správnosť metód a dát je v mnohých prípadoch overovaná vizuálne - používateľ dokáže overiť, či skutočne nasnímal to čo chcel a či neurobil chybu. Taktiež takto dokážeme overovať, či naše metódy pracujú správne a či nenastala chyba. Ďalším dôvodom pre zlepšenie vizualizéru je naša tímová prezentácia a celková prezentácia produktu. Pri prezentácií je práve vizualizér náš jediný výstup, ktorý vníma pozorovateľ. Ten nemusí mať s problematikou rekonštruovania 3d scény a v podstate ani s grafikou či informatikou ako takou žiadne skúsenosti. A práve vizualizér je jediná vec, prostredníctvom ktorej môže tento pozorovateľ a teoreticky budúci „zákazník“ s naším projektom prísť do kontaktu. A práve dobre vypracovaný vizualizér je preto dôležitou časťou nášho projektu a našej prezentácie.

5.4.2 NÁVRH

V našej práci sa zameriavame hlavne na implementáciu intuitívneho pohybu pomocou myši. Ďalšou navrhnutou metódou pohybu je ovládanie pomocou 3D myši. Pre správnu a hlavne intuitívnu implementáciu pohybu v priestore je nutné navrhnúť interakciu s vizualizérom tak aby bola priateľská pre používateľa a ľahko naučiteľná. Samotné dáta, ktoré vizualizujeme sú pripravené na prezentáciu implementovanými metódami segmentácie.

Nový vizualizér, ktorý je založený na vstavanom vizualizéri PCL knižnice, spočíva prevažne v implementácii vlastného ovládania (správne reakcie na pohyb myšou, správne interpretované klávesové vstupy do pohybov kamery vo vizualizéri) pre vizualizér a vyvinutie metód na spracovania vstupov z 3D myši. Okrem tohto hlavného problému vizualizéru sa tiež venujeme správnemu vkladaniu dát do priestoru v ktorom sa kamera pohybuje. Štartovacia pozícia kamery a jej orientácie v priestore je tiež veľmi dôležitá pre zorientovanie používateľa.

Pre používateľa je veľmi dôležité aby videl ako sa pomocou segmentačných metód mení používaný Point Cloud. Keď sa bude môcť prepínať medzi pôvodným a výsledným Point Cloudom dokáže identifikovať problémy metódy a uvidí čo zvolená metóda vlastne urobila.

5.4.3 IMPLEMENTÁCIA

Implementácia vizualizéru je riešená prostredníctvom funkcií z PCL knižnice. Z GUI sa zavolá metóda, ktorá vytvorí prázdne okno a nastaví prvotné parametre ako napr. farba pozadia. Druhá metóda toto okno naplní podľa zvolenej metódy z GUI a nastaví inicializačné parametre pre kameru na základe stredu pointcloudu, veľkosť bodov, ich farbu a zavolá triedu kde je definovaná interakcia pre okno vizalizéru. Vložený pointcloud je potom možné ľubovoľne prezerať v okne. V rámci interakcie je implementovaná aj metóda, ktorá povoľuje za behu zmeniť obsah okna a pozrieť si rozdiel spracovaného a pôvodného pointcloudu. Vieme vizualizovať všetky výsledky našich metód, vysegmentované plochy a ich hrany alebo ofarbený Point Cloud, ktorý je výstupom Birkis metódy.

Vizualizér obsahuje aj implementáciu pre interakciu pomocou 3D zariadenia SpaceNavigator, prípadne iného od rovnakej spoločnosti. Interakcia v okne je vykonávaná na základe koordinátov získavaných pri pohybe myšou. Podľa nich sa vykonávajú rôzne metódy pohybu kamery. Všetky tieto metódy sme implementovali sami, nakoľko knižnica PCL ani žiadna iná neobsahuje požadované metódy. Taktiež sme implementovali určitý spôsob akcelerácie. To znamená, že čím viac používateľ myš nakloní, tým rýchlejší bude pohyb. Či už pri interakcií myšou, 3D myšou alebo klávesnicou, vždy sme vychádzali z metód definovaných v knižnici PCL.

5.4.4 TESTOVANIE

Vizualizér sme testovali na používateľoch. Prvá fáza testovania bolo testovanie s tímom. Všetci členovia tímu postupne otestovali vizualizér, pričom sme vždy zapísali všetky poznámky a postupne

sa odstraňovali chyby. Keď už tím nemal väčšie výhrady, skúšali sme testovať vizualizér na nezainteresovaných osobách a celý postup sme opakovali.

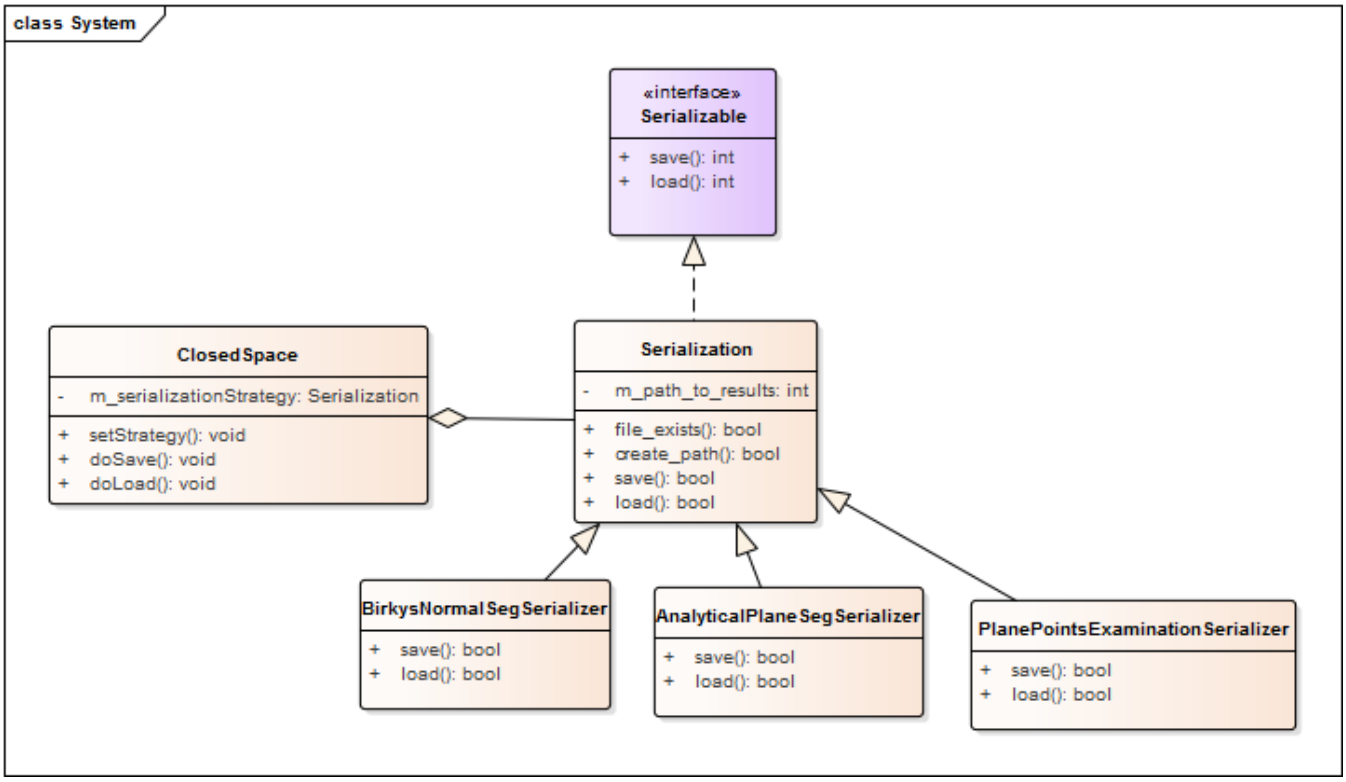
5.5 UKLADANIE MEDZIVÝSLEDKOV

5.5.1 ANALÝZA

Pri implementácii jednotlivých segmentačných algoritmov je častým problémom samotná výpočtová zložitosť. Avšak pri úpravách určitých menších krokov v celom algoritme (napr. parametrizácia) alebo doplnenia určitých krokov, pričom pôvodné kroky ostávajú nezmenené, nie je potrebné prepočítavať celý algoritmus znovu. Pri reálnych datasetoch (napr. apartmán) môže byť prepočítanie celej segmentačné metódy časovo náročné, čo v konečnom dôsledku veľmi spomaľuje prototypovanie. Za účelom urýchlenia prototypovania sme sa rozhodli pre serializáciu objektov, ktoré reprezentujú segmentačné metódy - *BirkysNormalSegmentation, AnalyticalPlaneSegmentation, PlanePointsExamination*.

5.5.2 NÁVRH

Serializáciu sme navrhli s využitím návrhového vzoru *Strategy*, pričom klientom sa stáva trieda *ClosedSpace*, ktorá si zvolí stratégiu podľa toho akú metódu vykonáva. Následne iba pred vykonávaním metódy *calculate* vyvolá metódu *load*, ktorá načíta dostupné uložené výsledky (ak žiadne nie sú, tak sa žiadne nenačítajú), ďalej vyvolá metódu *calculate* a nakoniec metódu *save*, ktorá uloží vypočítané medzivýsledky. Týmto sa vlastne uložia všetko medzivýsledky naprieč celým algoritmom, ak teda pridáme nový výpočet do algoritmu, stačí keď sa počíta iba ten a tie pôvodné sa len načítajú z disku.



5.5.3 IMPLEMENTÁCIA

Pri implementácií sme využili serializáciu pomocou knižnice *boost* a *io,* ktoré ponúka samotné *pcl*. V prípade ak je možné objekt uložiť ako typ *.pcd* (point cloud), tak sa uloží s využitím *io* od *pcl*. Ak to však možné nie je (napr. *m_clusters* - vektor typu *PointIndices*) je potrebné zabezpečiť serializáciu. Túto sme implementovali pomocou knižnice *boost* s využitím triedy *Archive*.

Jednotlivé súbory sú ukladané v priečinku, ktorý nesie meno po názve datasetu. Nachádzajú sa v *QT_GUI* v priečinku *<meno_datasetu_bezpripony>/normalSeg/<typ_metody>/*. Ak táto cesta neexistuje systém ju automaticky vytvorí a následne do nej ukladá jednotlivé medzivýsledky. Následne pri načítaní sa hľadajú medzivýsledky v tejto ceste. Ak medzivýsledok neexistuje, tak sa nenačíta. Celý tento proces zabezpečuje trieda *Serialization*.

Implementovaná je serializácia segmentačnej metódy pomocou normál (*BirkysNormalSegSerializer*). Táto slúži ako šablóna pre implementáciu ďalších metód *save/load* v rámci ostatných serializérov.

5.5.4 TESTOVANIE

Pri testovaní serializácie sme manuálne vyskúšali uloženie medzivýsledkov pre viaceré datasety. Následne sme skúsili segmentačnú metódu vypočítať znovu, tento krát už s uloženými medzivýsledkami. Ako akceptačné kritérium bolo úspešné vypočítanie metódy s načítaním medzivýsledkov a následne overenie výsledkov pomocou vizualizéra.

Jednotlivé chybové stavy sú ošetrené pomocou *try-catch* blokov v kóde.

6. INŠTALAČNÁ PRÍRUČKA

Inštalačná príručka je napísaní pre Visual Studio 2013

Po stiahnutí repozitára je potrebné nainštalovať nasledovné súčasti.

6.1 NÁVOD NA INŠTALÁCIU PCL 1.8

Nastaviť systémové premenné: (môžu sa líšiť)

BOOST_ROOTC:\Program Files\PCL 1.7.2\3rdParty\Boost

EIGEN_ROOT C:\Program Files\PCL 1.7.2\3rdParty\Eigen\eigen3

FLANN_ROOT C:\Program Files\PCL 1.7.2\3rdParty\FLANN

QHULL_ROOT C:\Program Files\PCL 1.7.2\3rdParty\Qhull

VTK_ROOT C:\Program Files\PCL 1.7.2\3rdParty\VTK

Nainštalovať:

CMAKE https://cmake.org/files/v3.8/cmake-3.8.0-rc1-win64-x64.msi

Cuda 8.0 https://developer.nvidia.com/cuda-downloads

Stiahnuť:

PCL 1.8 https://github.com/PointCloudLibrary/pcl

1. Rozbaliť pcl-master adresár
2. Vytvoriť podzložku build v adresári pcl-master
3. Otvoriť cmake GUI
4. Nastaviť zložku “source code” na rozbalený adresár plc
5. Nastaviť zložku “binaries” na vytvorený podadresár build
6. Ak nenájde VTK tak treba zadať cestu manuálne
   C:/Program Files/PCL 1.7.2/3rdParty/VTK/lib/cmake/vtk-6.2
7. Generate
8. Otvoriť visual studio ako administrátor
9. Build projektu install v debug a v release

6.2 NÁVOD NA INŠTALÁCIU KINECT / KINFU

Nainštalovať (dodržať poradie):

Kinect SDK 2.0 https://www.microsoft.com/en-us/download/details.aspx?id=44561

Cuda 8.0 https://developer.nvidia.com/cuda-downloads

OpenNI2 https://s3.amazonaws.com/com.occipital.openni/OpenNI-Windows-x64-2.2.0.33.zip

OpenNI https://structure.io/openni

CMAKE https://cmake.org/files/v3.8/cmake-3.8.0-rc1-win64-x64.msi

Nastaviť systémové premenné: (možu sa líšiť)

BOOST_ROOT C:\Program Files\PCL 1.7.2\3rdParty\Boost

EIGEN_ROOT C:\Program Files\PCL 1.7.2\3rdParty\Eigen\eigen3

FLANN_ROOT C:\Program Files\PCL 1.7.2\3rdParty\FLANN

QHULL_ROOT C:\Program Files\PCL 1.7.2\3rdParty\Qhull

VTK_ROOT C:\Program Files\PCL 1.7.2\3rdParty\VTK

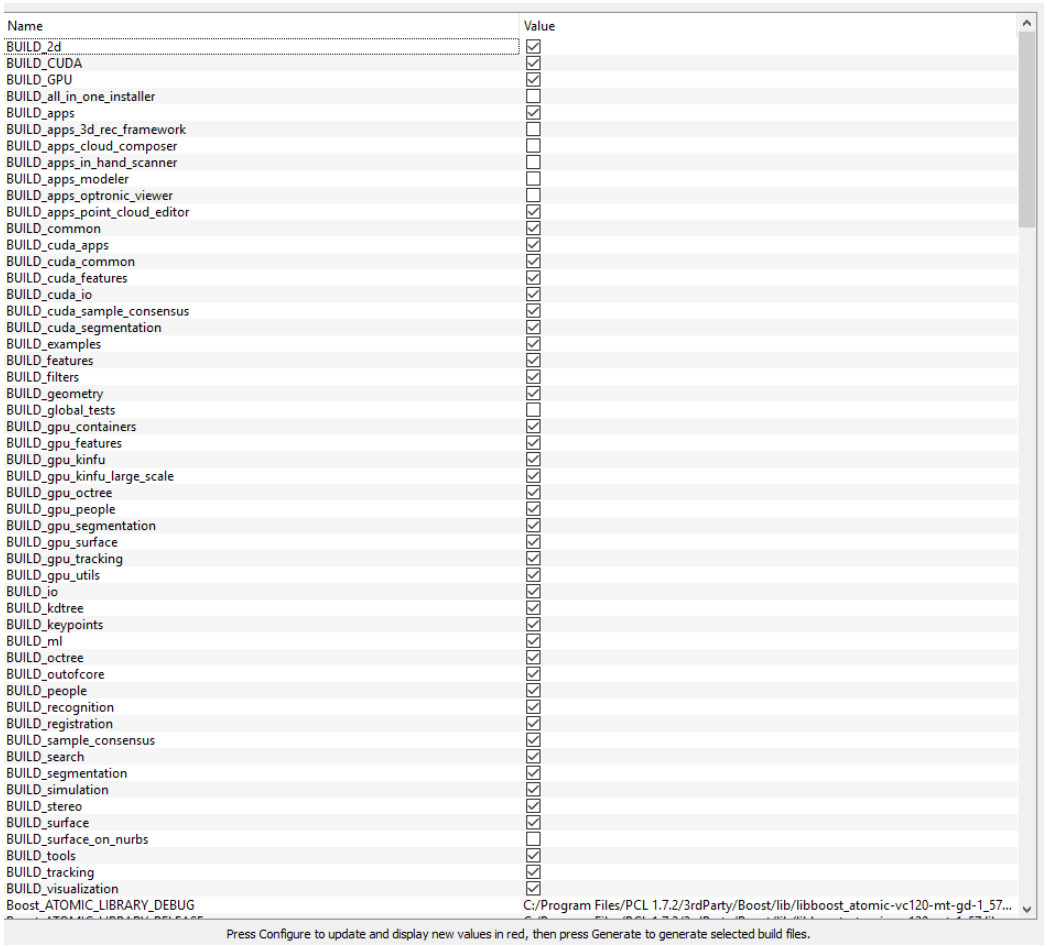
CUDA_PATH C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0

premenné OPENNI a OPENNI2 by sa mali vytvoriť sami

Stiahnuť:

PCL 1.8 https://github.com/PointCloudLibrary/pcl

1. Rozbaliť pcl 1.8 release
2. Vytvoriť podzložku build v adresári pcl-master
3. Otvoriť cmake GUI
4. Nastaviť zložku “source code” na rozbalený adresár plc
5. Nastaviť zložku “binaries” na vytvorený podadresár build
6. configure -> visual studio 2013 WIN64
7. Ak nenájde VTK tak treba zadať cestu manuálne
   C:/Program Files/PCL 1.7.2/3rdParty/VTK/lib/cmake/vtk-6.2



8.
9. opakovať configure až kým nevyhlási žiaden error
10. Generate
11. Otvoriť visual studio ako administrator
12. Build projektu install v debug a v release
13. vytvoriť PCL_ROOT_18 na C:\Program Files\PCL\
14. pre Kinect_GUI prestaviť cestu v configuration properties/debugging/enviromet

PATH="$(QTDIR)\bin%3b$(QTDIR)\bin%3b"$(QTDIR)\bin%3b$(QTDIR)\bin%3b"$(QTDIR)\bin%3b$(QTDIR)\bin%3b$(VTK_ROOT)\bin%3b$(QHULL_ROOT)\bin%3b$(FLANN_ROOT)\bin%3b$(BOOST_ROOT)\bin%3b$(PCL_ROOT_18)\bin%3b$(OPENNI2_REDIST64)%3b$(SGCORE_ROOT)%3b$(KFUSION_ROOT)\bin%3b$(PATH)

v prípade erroru v cuda projekte opraviť podľa https://github.com/PointCloudLibrary/pcl/issues/1674

6.3 NÁVOD NA INŠTALÁCIU 3DCONNEXION MYŠI

1. Stiahnuť driver pre myš z: http://www.3dconnexion.eu/service/drivers.html
   Môže vyžadovať registráciu ako developer (Teraz som to skúšal a nechcelo ju tak si nie som istý kedy ju to chce)
2. Nainštalovať ho
3. Nastaviť systémovú premennú prostredia:
   Connex_ROOT C:\Program Files (x86)\3Dconnexion
   Záleží kde ste nainštalovali driver, teda môžete potrebovať inú !!
   Po nastavení treba reštartovať Visual Studio, ak je zapnuté
4. V projekte *Visualization* -> Debugging -> Environment pridať do PATH vytvorenú premennú
5. V projekte *Visualization* -> *VC++ Directories* -> *Include Directorie*s pridať
   $(Connex_ROOT)\3DxWare SDK\Inc
6. V projekte *Visualization* -> *VC++ Directories* -> *Library Directorie*s pridať
   $(Connex_ROOT)\3DxWare SDK\Lib\x64
7. V projekte *QT_GUI* -> *VC++ Directories* -> *Include Directorie*s pridať
   $(Connex_ROOT)\3DxWare SDK\Inc
8. V projekte *QT_GUI* -> *VC++ Directories* -> *Library Directorie*s pridať
   $(Connex_ROOT)\3DxWare SDK\Lib\x64
9. V projekte *QT_GUI* -> *Linker* -> *Additional* Dependencies pridať **siapp.lib**

Takto nastavené by to malo bez problémov fungovať.

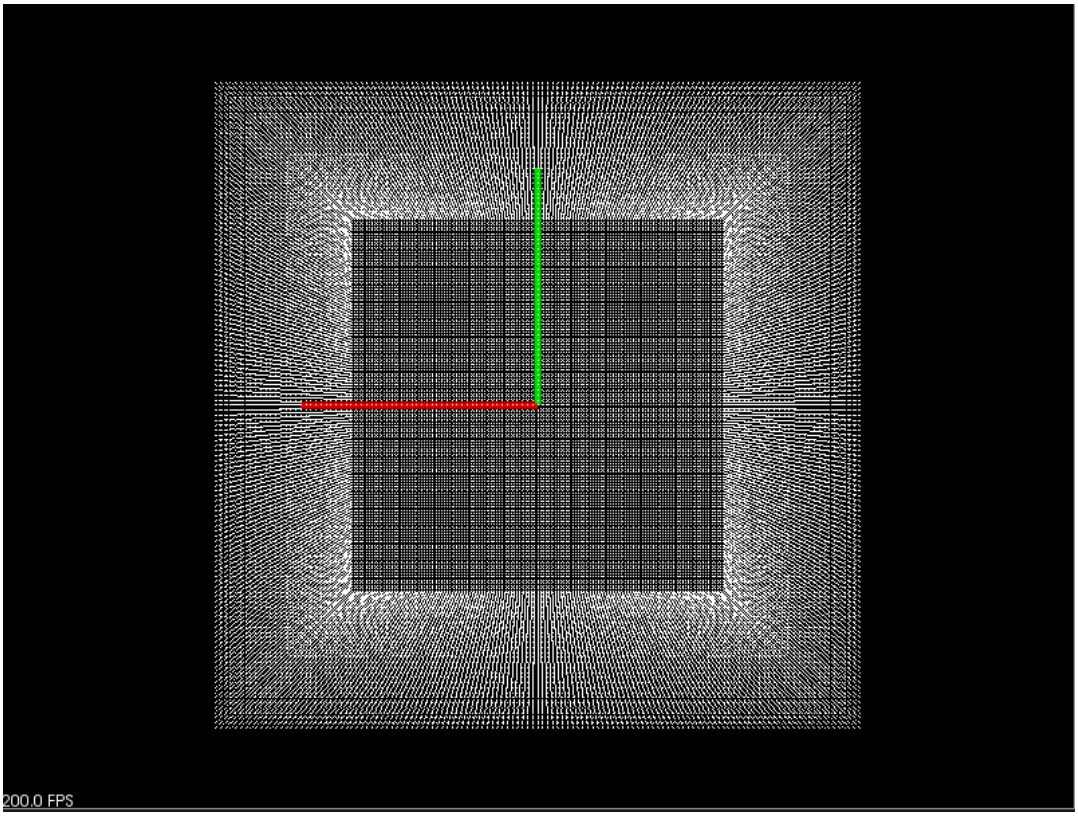
6.4 ĎALŠIE ZÁVISLOSTI

K projektu je taktiež potrebné doinštalovať knižnicu TinyXML 2.6.2 , OpenCV (premenná OPENCV_ROOT) a SgCore (premenná SGCORE_ROOT). V prípade, že sa nebude využívať kinect je možné nahradiť PCL 1.8 verziou 1.7.2

7. TECHNICKÁ DOKUMENTÁCIA

7.1 MANUÁL K OVLÁDANIU VIZUALIZÉRA

Základná poloha

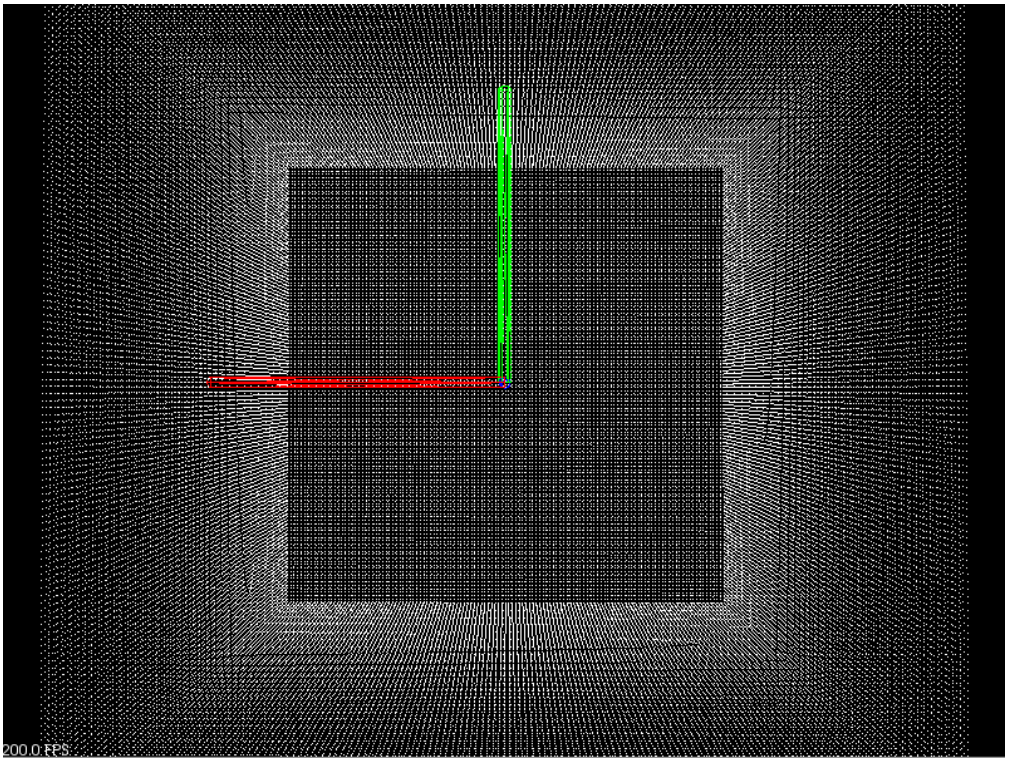


W	Priblíženie
S	Oddialenie
A	Pohyb doľava
D	Pohyb doprava
8	Pohyb kamerou nahor
2	Pohyb kamerou nadol
4	Pohyb kamerou doľava
6	Pohyb kamerou doprava
5	Rotácia nahor
0	Rotácia nadol

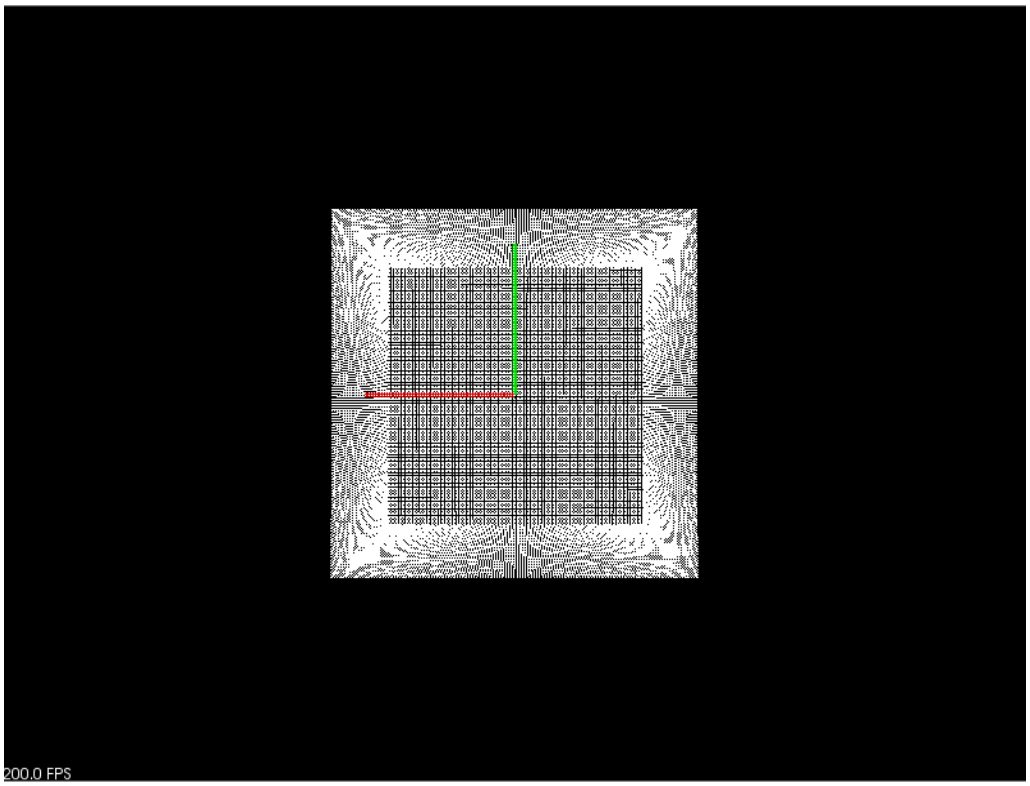
1	Rotácia doľava
3	Rotácia doprava
7	Náklon doľava
9	Náklon doprava

Možnosti pohybu v priestore

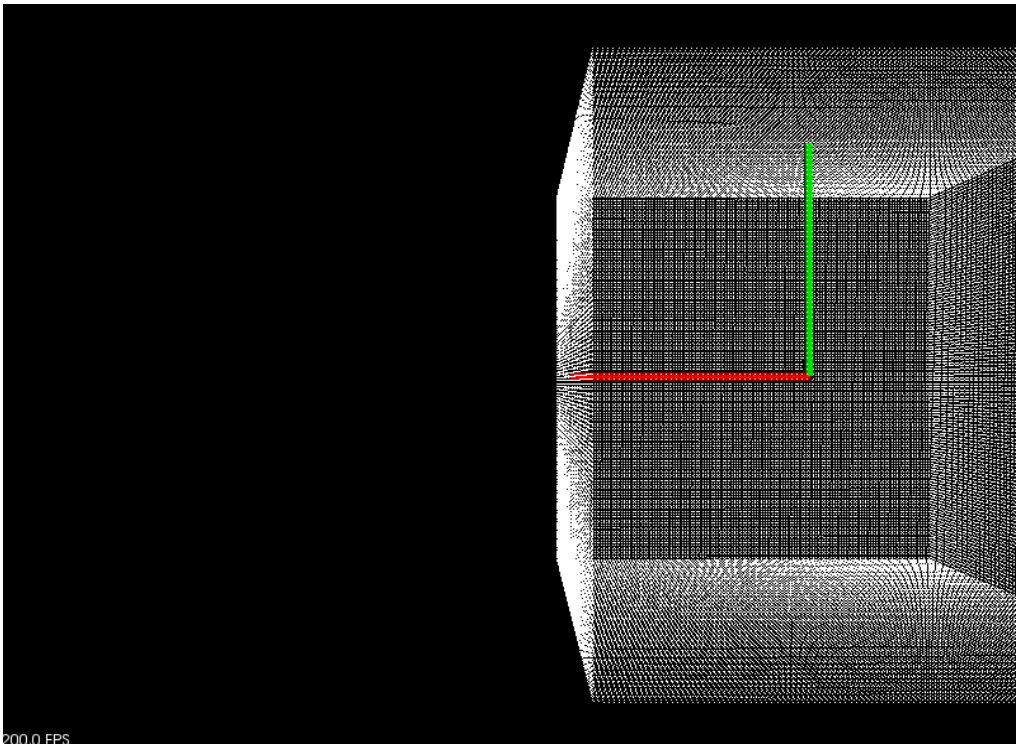
W / Mouse Wheel - Priblíženie



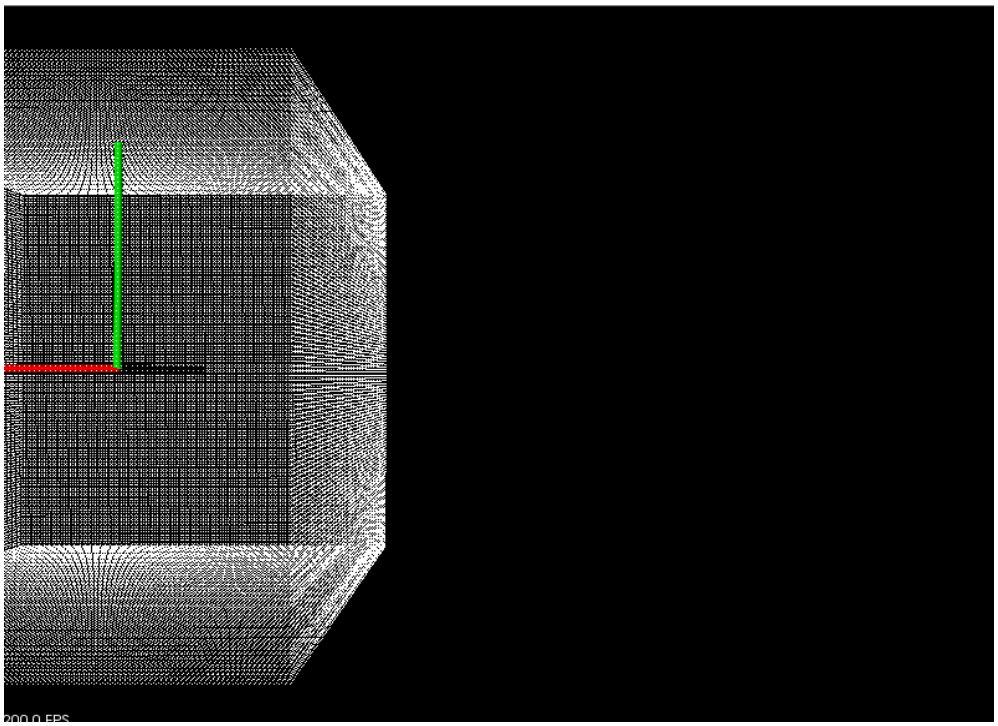
S / Mouse Wheel - Oddialenie



A – Pohyb doľava

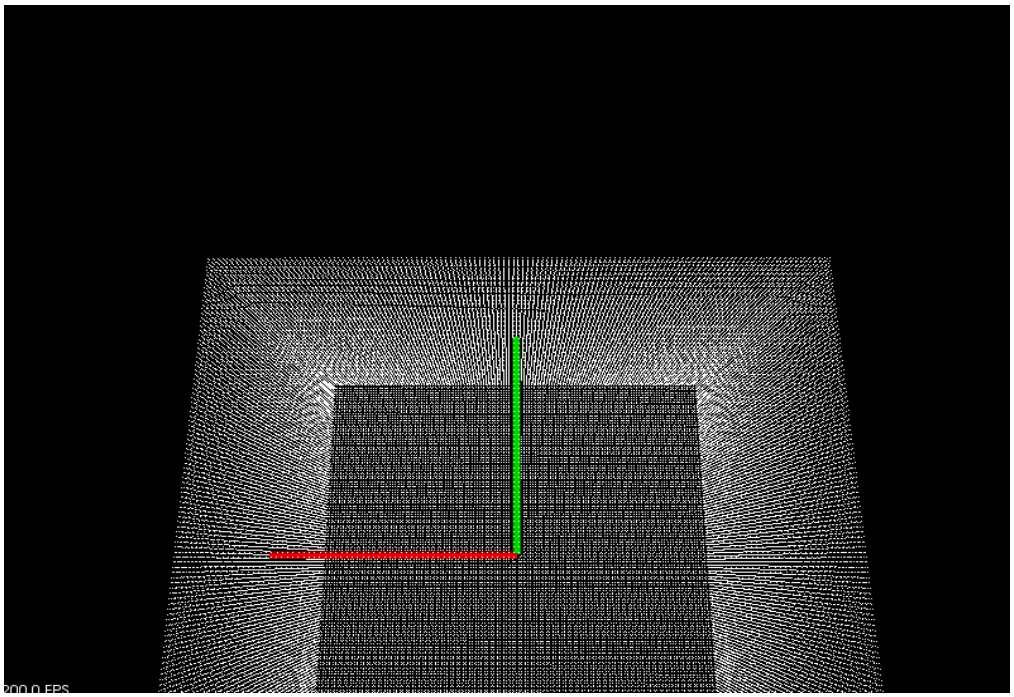


D – Pohyb doprava

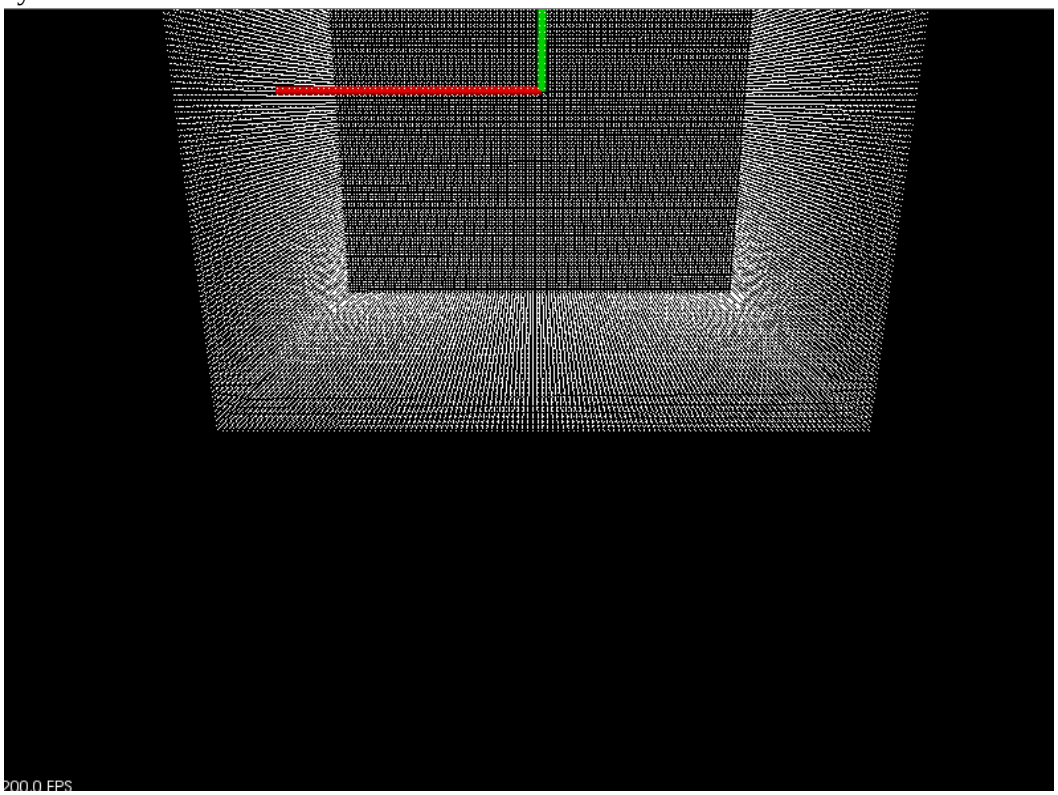


Rotovanie kamery

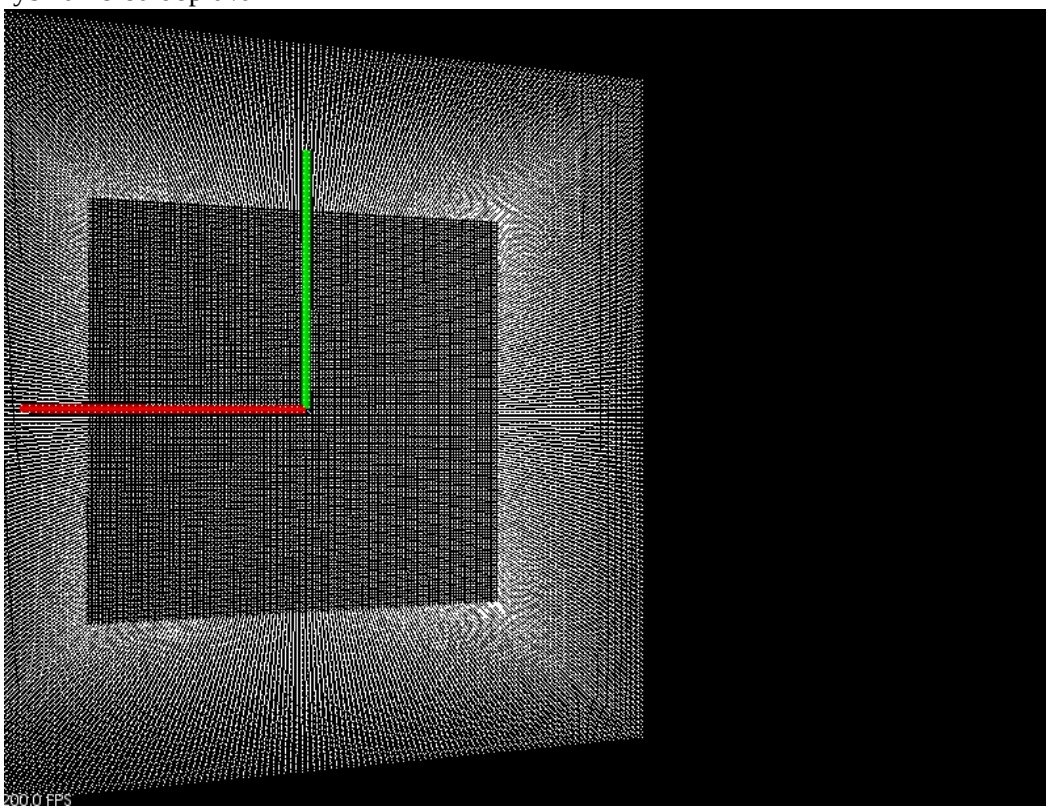
8 – Pohyb kamerou nahor



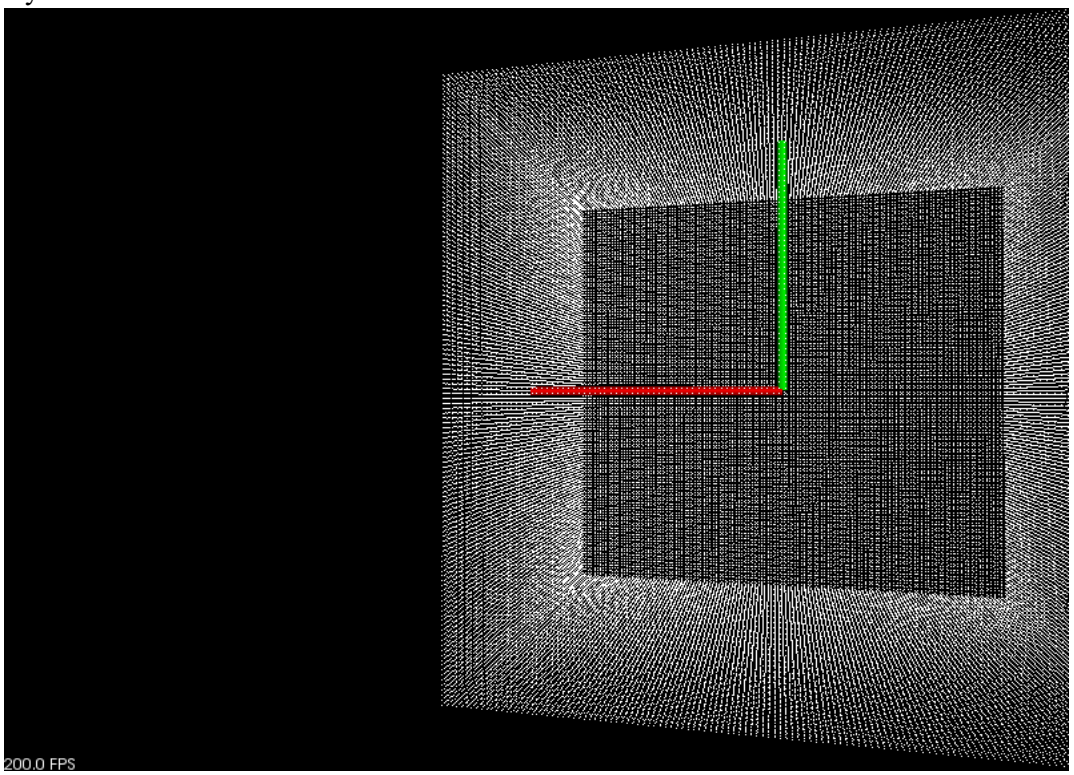
2 – Pohyb kamerou nadol



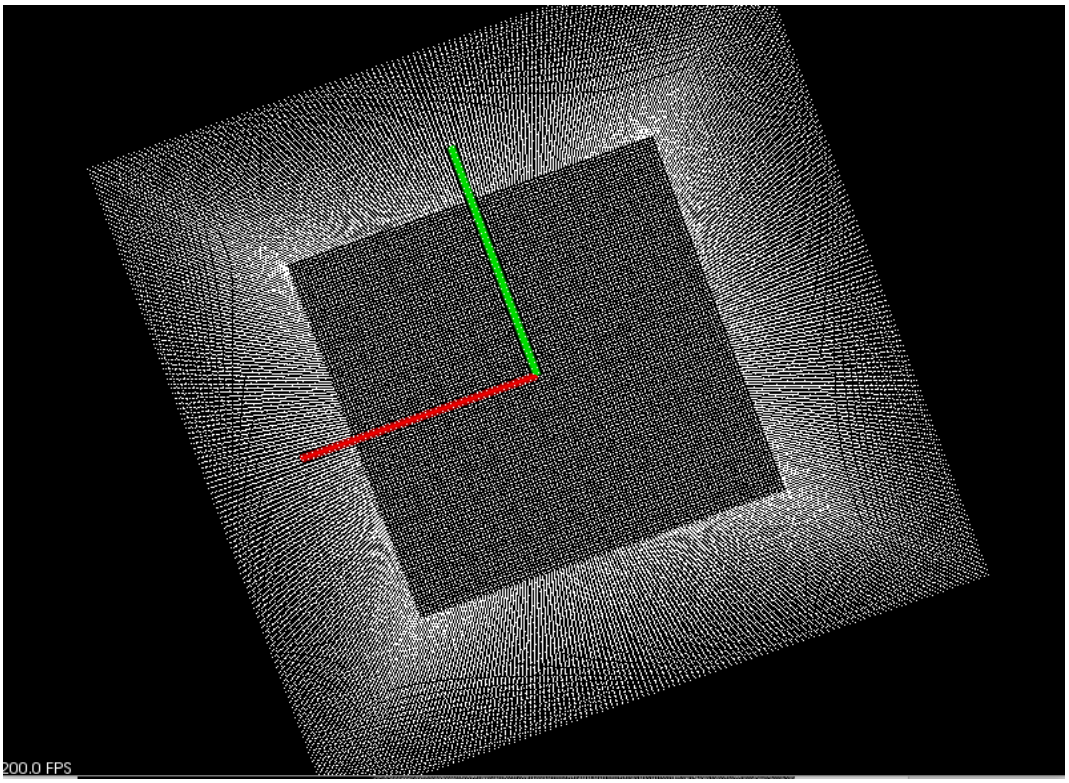
6 – Pohyb kamerou doprava



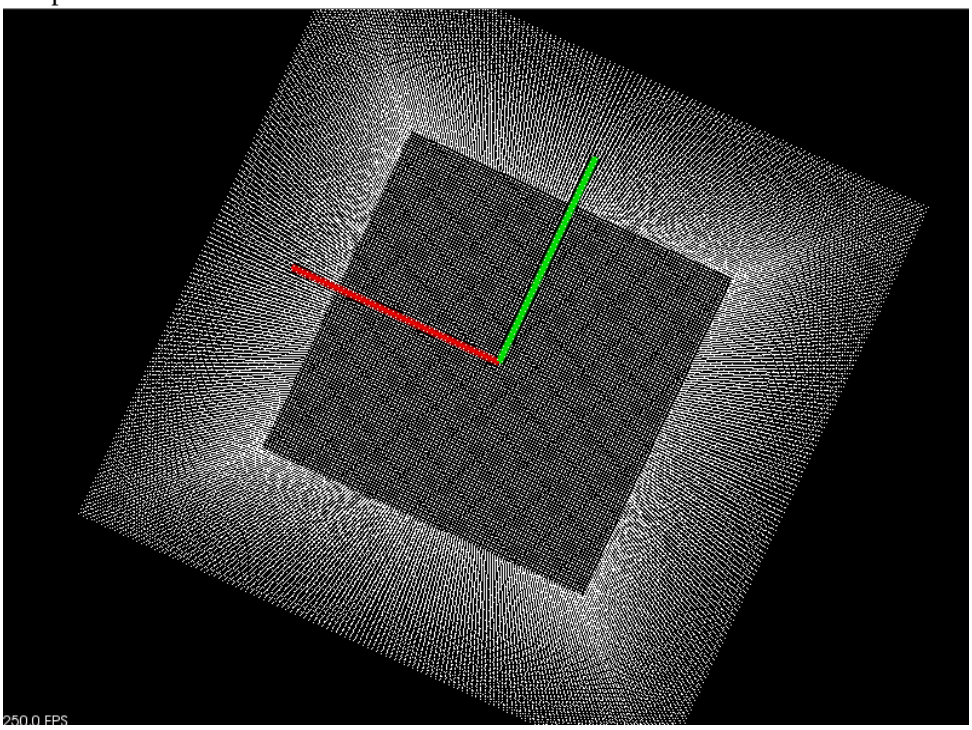
4 – Pohyb kamerou doľava



7 – Náklon doľava

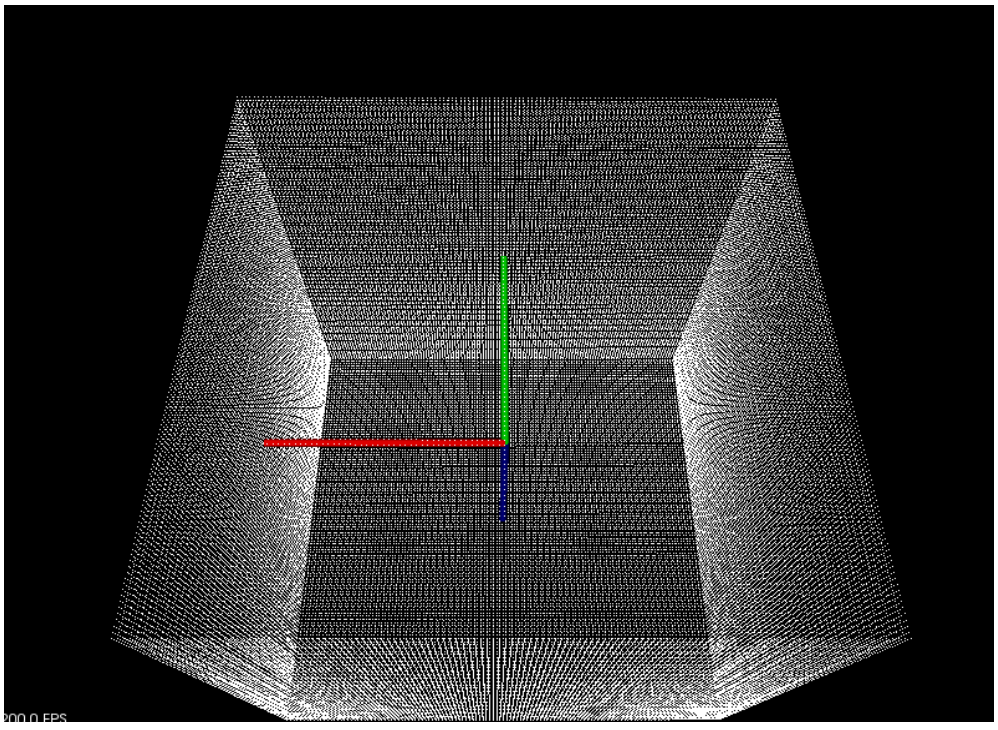


9 – Náklon doprava

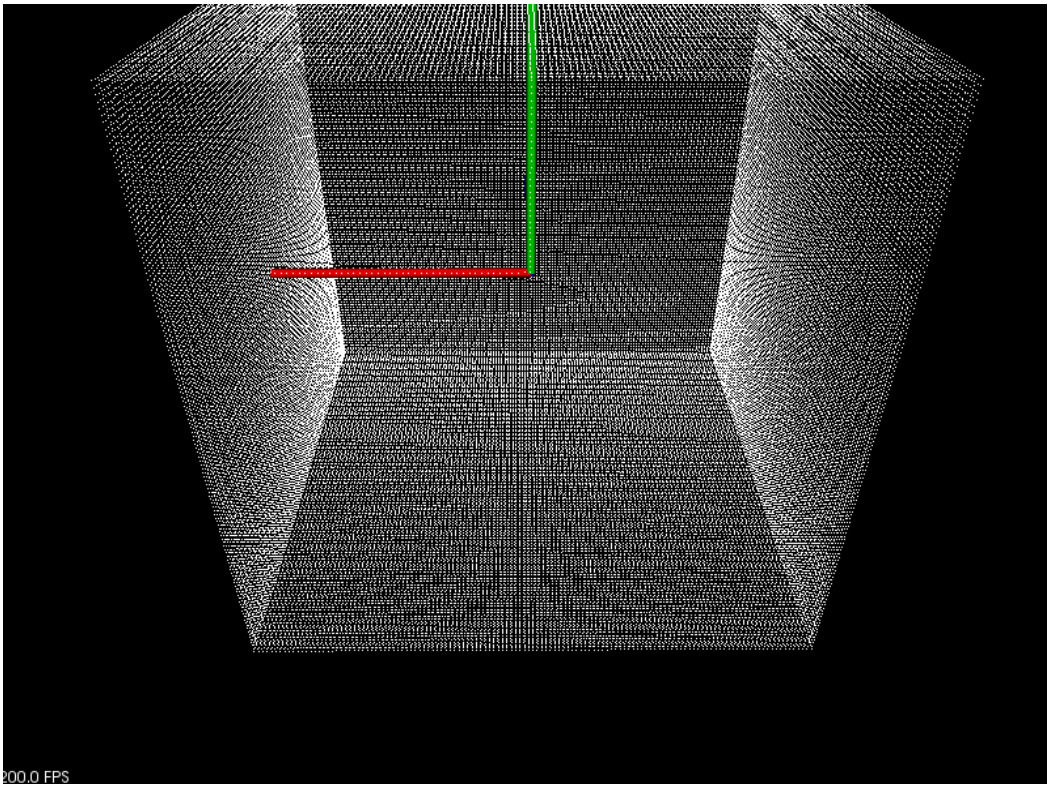


Pohybovanie „objektom“

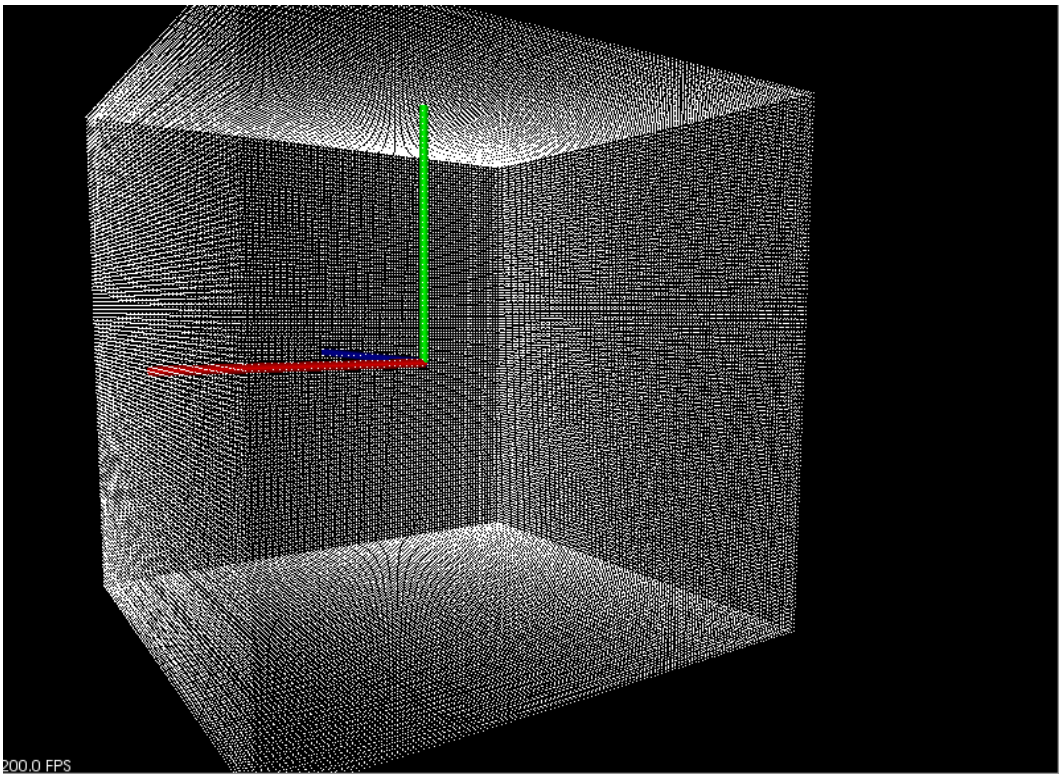
5 – Rotácia nahor



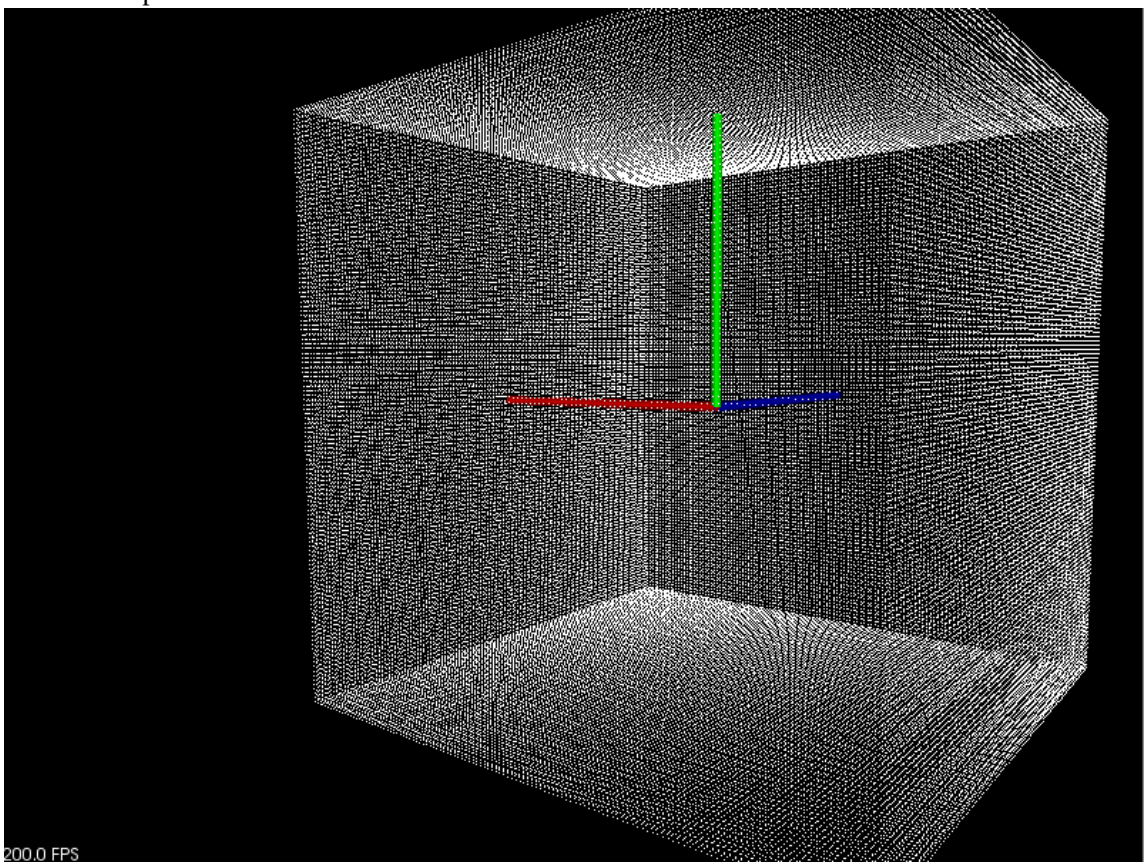
0 – Rotácia nadol



1 – Rotácia doľava



3 – Rotácia doprava



Ďalšie funkcie

r	Resetovanie polohy kamery
+/-	Zväčšovanie/Zmenšovanie veľkosti bodov
g	Zobrazenie mierky
Ctrl + s	Uloženie parametrov kamery
Ctrl + r	Obnovenie parametrov kamery (vopred uložených)
Alt + s	Vypnutie/zapnutie stereo módu zobrazenia
Alt + f	Vypnutie/Zapnutie max veľkosti okna
f	Mód lietania k bodom