

**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU  
ELEKTROTEHNIČKI FAKULTET**

**Sveučilišni studij**

**IZGRADNJA 3D MODELΑ SCENE POMOĆU 3D  
KAMERE**

**Diplomski rad**

**Marijan Svalina**

**Osijek, 2013.**

# Sadržaj

<b>1. Uvod</b>	<b>1</b>
1.1. Zadatak diplomskog rada	2
<b>2. Pregled korištenih tehnologija i algoritama</b>	<b>3</b>
2.1. Microsof Kineckt 3D kamera	3
2.2. ROS biblioteka i alati	3
2.3. Biblioteka Pointcloud	3
2.4. Istovremena lokalizacija i mapiranje	3
2.5. Poisson algoritam za rekonstrukciju površine	4
<b>3. Izgradnja 3D modela scene</b>	<b>8</b>
3.1. Snimanje scene 3D kamerom i RGBDSlam programom	8
3.2. Izgradnja 3D modela scene pomoću mreže trokuta	12
<b>4. Rezultati</b>	<b>21</b>
<b>5. Zaključak</b>	<b>22</b>
<b>Literatura</b>	<b>23</b>
<b>Sažetak</b>	<b>25</b>
<b>Životopis</b>	<b>26</b>
<b>Prilozi</b>	<b>27</b>

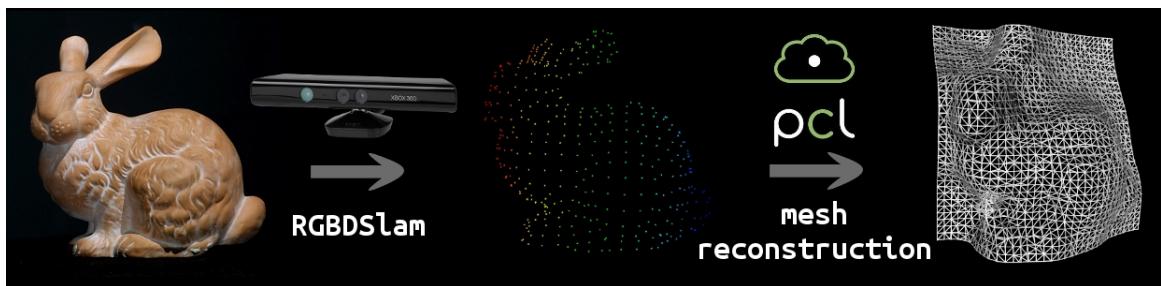
# 1. UVOD

Pojava i dostupnost jeftinog i kvalitetnog 3D senzora Microsoft Kinect kamere 2010. godine za igračku konzolu Xbox 360 uvelike je doprinjela razvoju računalnog vida. Senzor u VGA rezoluciji frekvencijom od 30Hz daje informacije o dubini i sliku u boji. Time omogućava da se na prirodniji način pristupi rješavanju osnovnih problema računalnog vida. Upravo to se i dogodilo te su znanstvenici, programeri i hakeri razvili upravljačke programa, alate i algoritme za korištenje Kineckt senzora i sličnih uređaja za prikupljanje oblaka točaka. Većina kreiranog softvera je objavljena pod slobodnim licencama koje omogućavaju slobodu upotrebe programa u bilo koje svrhe, slobodu proučavanja i primjenjivanja stečenog znanja, slobodu distribuiranja kopija u cijelosti ili u dijelovima te slobodu mjenjanja, poboljšavanja i distribuiranje derivacijskih programa. Upravo ti programi su postavili temlje za ovaj diplomski rad.

Rad se sastoji iz tri dijela. Prvi dio odnosi se na pregled korištenih tehnologija i algoritama. **\*\*Napisati jos koju recenicu o tehnologijama i algoritmima\*\***. Drugi dio je praktični dio i govori o izgradnji 3D modela scene. Prvo je objašnjen postupak snimanja scene 3D kamerom i RGBDSlam programom, a zatim je dan pregled izgradnje 3D modela scene mrežom trokuta. Treći dio rada prikazuje rezultate snimanja i izrade te ispituje funkcionalnost i kvalitetu postupka.

## 1.1. Zadatak diplomskog rada

Program RGBDSLAM raspoloživ u okviru incijative dijeljenja algoritama OpenSLAM omogućava izgradnju 3D modela objekata i scena pomoću 3D kamere. Razviti program za izgradnju 3D modela u obliku mreže trokuta koristeći biblioteku PointCloud. Kombinacijom ova dva programa mogu se izgraditi 3D modeli objekata i scena snimljenih iz više pogleda. Zadatak je ispitati funkcionalnost navedenog postupka kao i kvalitetu dobivenog rezultata izgradnjom nekoliko 3D modela objekata i scena.



Slika 1.1.: Grafički prikaz projekta upotrebom Standfordovog zeca<sup>1</sup>

---

<sup>1</sup>Standford Bunny 3D model su originalno konstruirali 1994 Greg Turk i Marc Levoy i od tada je postao najčešće upotrebljевani model za testiranje tehnika u računalnoj grafici. <http://www.gvu.gatech.edu/people/faculty/greg.turk/bunny/bunny.html>

## **2. PREGLED KORIŠTENIH TEHNOLOGIJA I ALGORITAMA**

- 2.1. Microsof Kineckt 3D kamera**
- 2.2. ROS biblioteka i alati**
- 2.3. Biblioteka Pointcloud**
- 2.4. Istovremena lokalizacija i mapiranje**

## 2.5. Poisson algoritam za rekonstrukciju površine

Poisson algoritam za rekonstrukciju površine [7] razvijen je suradnjom Michaela Kazhdana i Matthewa Bolitha s Johns Hopkins sveučilišta u Baltimoru i Huguesa Hoppea iz Microsoft Researcha u Redmondu. Također Kazhdan i Bolitho su implementirali<sup>1</sup> Poisson algoritam i objavili kod pod BSD licencom. Na osnovu tog rada algoritam je dodan i u PCL biblioteku.

U ovom potpoglavlju nalazi se osnovna ideja i kratak matematički pregled algoritma. Opisano je ograničenje algoritma te parametri kojima se može upravljati rekonstrukcija površine.

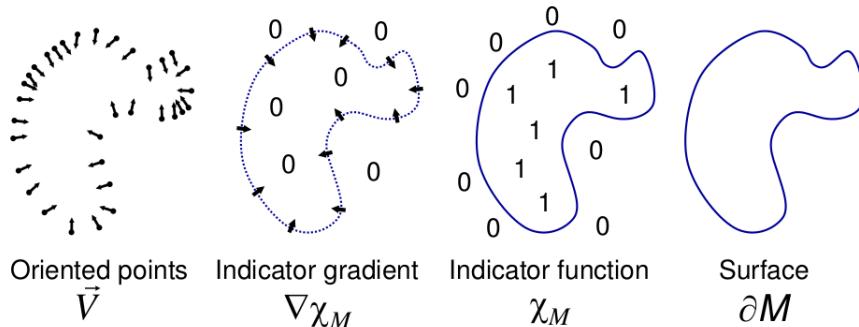
---

<sup>1</sup>Originalna implementacija Poisson algoritma se nalazi na <http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version5.5/>

### 2.5.1. Osnovna ideja i kratak matematički pregled

Poisson algoritam pristupa problemu rekonstrukcije površine rješavanjem Poissonove jednadžbe. To čini upotrebom metode implicitne funkcije. Točnije računanjem 3D indikacijske funkcije  $\chi$  (definiranom s 1 u točkama unutar modela, odnosno s 0 u točkama izvan) i dohvaćanjem rekonstruktruirane površine izvlačenjem odgovaraajuće iso-površine.

Algoritam se oslanja na ideju da postoji cijelovita veza između orijentiranih normala uzetih s površine modela i indikacijske funkcije modela. Točnije, gradijent indikacijske funkcije je polje vektora koje je uglavnom popunjeno nulama (jer je indikacijska funkcija uglavnom konstantna), osim kod točaka blizu površine gdje je jednako unutrašnjim normalama površine. Stoga, uzorci orijentiranih normala mogu biti promatrani kao gradijent modelove indikacijske funkcije kao što je prikazano na slici 2.1.



Slika 2.1.: Prikaz Poisson rekonstrukcije u 2D, izvor: [7]

Problem računanja indikacijske funkcije se svodi na invertiranje operatora gradijenta, odnosno pronalazak funkcije skalara  $\chi$  čiji gradijent najbolje aproksimira polje vektora  $\vec{V}$  definirano uzorcima, odnosno

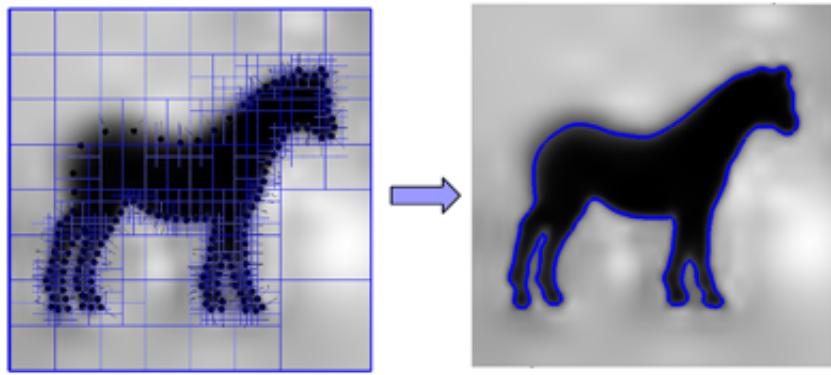
$$\min_{\chi} \|\nabla \chi - \vec{V}\|.$$

Ako se primjeni operator divergencije, tada se taj problem pretvara u standardni Poissonov problem: računanje funkcije skalara  $\chi$  čiji laplasijan (divergencija gradijenta) je jednak divergenciji polja vektora  $\vec{V}$ ,

$$\Delta \chi \equiv \nabla \cdot \nabla \chi = \nabla \cdot \vec{V}.$$

Predstavljanje rekonstrukciju površine kao Poissonov problem pruža nekoliko prednosti. Mnoge implicitne metode namještanja površina segmentiraju podatke u regije za lokalno namještanje i onda te lokalne aproksimacije spajaju upotrebom funkcija stapanja. Za razliku od njih, Poisson rekonstrukcija je globalno rješenje koje razmatra sve podatke odjednom, bez upotrebe heurističkih podijela i stapanja. Zbog toga Poisson rekonstrukcija kreira izrazito glatku površinu koja robusno aproksimira šumovite podatke.

Za izvlačenje iso-površine Poisson algoritam koristi Marching Cubes algoritam [10] koji kreira octree strukturu podataka za prikaz površine. Kao što se vidi na slici 2.2. Marching Cubes algoritam dijeli oblak točaka u mrežu voxela marširajući kroz oblak i analizira koje točke čine iso-površinu objekta. Detektiranjem koji rubovi voxela presjecaju iso-površinu modela algoritam kreira mrežu trokuta. Više informacija o izvlačenju površine se mogu pronaći u radu "Unconstrained Isosurface Extraction on Arbitrary Octrees" Michaela Kahzdana [8].



Slika 2.2.: Prikaz Marching cubes algoritma, izvor: [8]

### 2.5.2. Ograničenje Poisson algoritma

Ograničenje implementacije Poisson algoritma je u tome što ne uzima u obzir informacije asocirane s načinom stjecanja oblaka točaka. Slika 2.3. pokazuje kip Bude i vidi se primjer takvog ograničenja. Budući da nema točaka između Budinih nogu, Poisson algoritam spaja te dvije regije. Algoritam se može unaprijediti ugradnjom dodatne informacije poput vidokruga i na taj način izbjegći to ograničenje.



Slika 2.3.: Rekonstrukcija modela “Happy Buddha” VRIP<sup>2</sup> algoritam (lijevo) i Poisson algoritma (desno), izvor: [7]

---

<sup>2</sup>VRIP - Volumetric Range Image Processing [3]

### 2.5.3. Parametri Poisson algoritma

Postoji nekoliko parametara koji utječu na rezultat rekonstrukcije.

- **Depth:** dubina octree stabla koje se koristi za rekonstrukciju. Zadana vrijednost 8.
- **SolverDivide:** postavlja dubinu kod kojeg bloka Gauss-Seidel metoda riješava Laplasovu jednadžbu. Zadana vrijednost 8.
- **IsoDivide:** postavlja dubinu kod kojeg bloka ekstraktor iso-površine izvlači iso-površinu. Zadana vrijednost 8.
- **SamplesPerNode:** postavlja minimalni broj točaka koje se trebaju nalaziti unutar octree čvora kako se octree konstrukcija prilagođava gustoći sempliranja. Za podatke bez šuma 1 - 5, sa šumom 15 - 20.
- **Scale:** omjer između promjera kocke korištne za rekonstrukciju i promjera kocke koja omeđuje uzorke. Zadana vrijednost 1.25.
- **Confidence:** postavljanje zastavice govori rekonstruktorciji da koristi veličinu normala kao informaciju o pouzdanosti. Ako nije postavljena sve normale se normaliziraju prije rekonstrukcije.

Od nabrojanih parametara najvažniji utjecaj na generiranu mrežu imaju **SamplesPerNode** i **Depth**. Veća dubina octree stabla rezultira većom precinosti mreže voxela jer Marching Cubes algoritam ulazi dublje u stablo. Manja dubina (između 5 i 7) daje glađi model ali s manje detalja. **SamplesPerNode** parametar definira koliko će točaka Marchin Cubes algoritma staviti u jedan čvor resultantnog octree stabla. Ako algoritam radi s podacima punim šuma velik uzorak točaka (15 - 20) po čvoru pruža glađenje ali se gube detalji. Dok rad s malim vrijednostima (1 - 5) održava razinu detalja visokom. Velike vrijednosti reduciraju kranji broj vrhova poligona, dok male održavaju broj vrhova visokim.

### 3. IZGRADNJA 3D MODELA SCENE

#### 3.1. Snimanje scene 3D kamerom i RGBDSlam programom

Program RGBDSlam omogućava izgradnju 3D modela objekata i scena u unutrašnjosti prostorija (oblak točaka u boji) rukom upravljanom kamerom tipa Kinect. Razvijen je suradnjom sveučilišta Albert-Ludwigs-Universität<sup>1</sup> u Freiburgu i Technische Universität München<sup>2</sup>. Slobodan je program objavljen pod GPLv3<sup>3</sup> licencom. Izvorni kod je dostupan na Google code<sup>4</sup> stranicama. U prilogu diplomskog rada nalaze se upute za prevođenje i instaliranje programa.

---

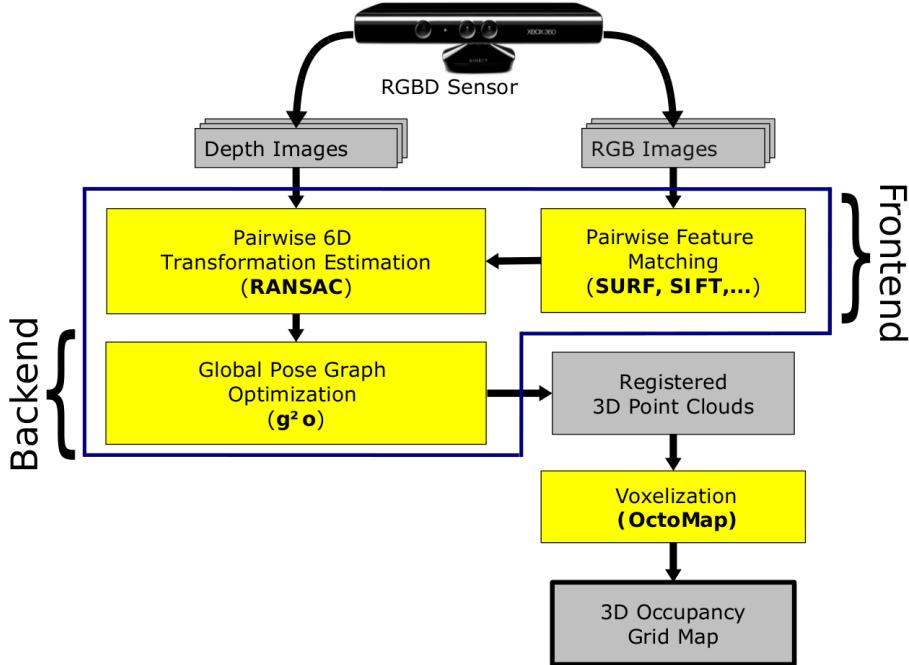
<sup>1</sup>Felix Endres i Juergen Hess sa odijela Autonomous Intelligent Systems koji vodi Prof. Dr. Wolfram Burgard.

<sup>2</sup>Nikolas Engelhard sa odijela Computer Vision Group koji vodi Dr. Juergen Sturm.

<sup>3</sup>GNU General Public License version 3 slobodna je licenca koja osigurava osnovna prava slobodnih programa. Pravo na korištenje, proučavanje, kopiranje i poboljšavanje. Izvor: <http://www.gnu.org/licenses/gpl-3.0.html>

<sup>4</sup>RGBDSlam program moguće je preuzeti sa svn programom sa stranice [http://alufr-ros-pkg.googlecode.com/svn/trunk/rbgd slam\\_freiburg/](http://alufr-ros-pkg.googlecode.com/svn/trunk/rbgd slam_freiburg/)

### 3.1.1. Sažet opis rada RGBDSlam programa



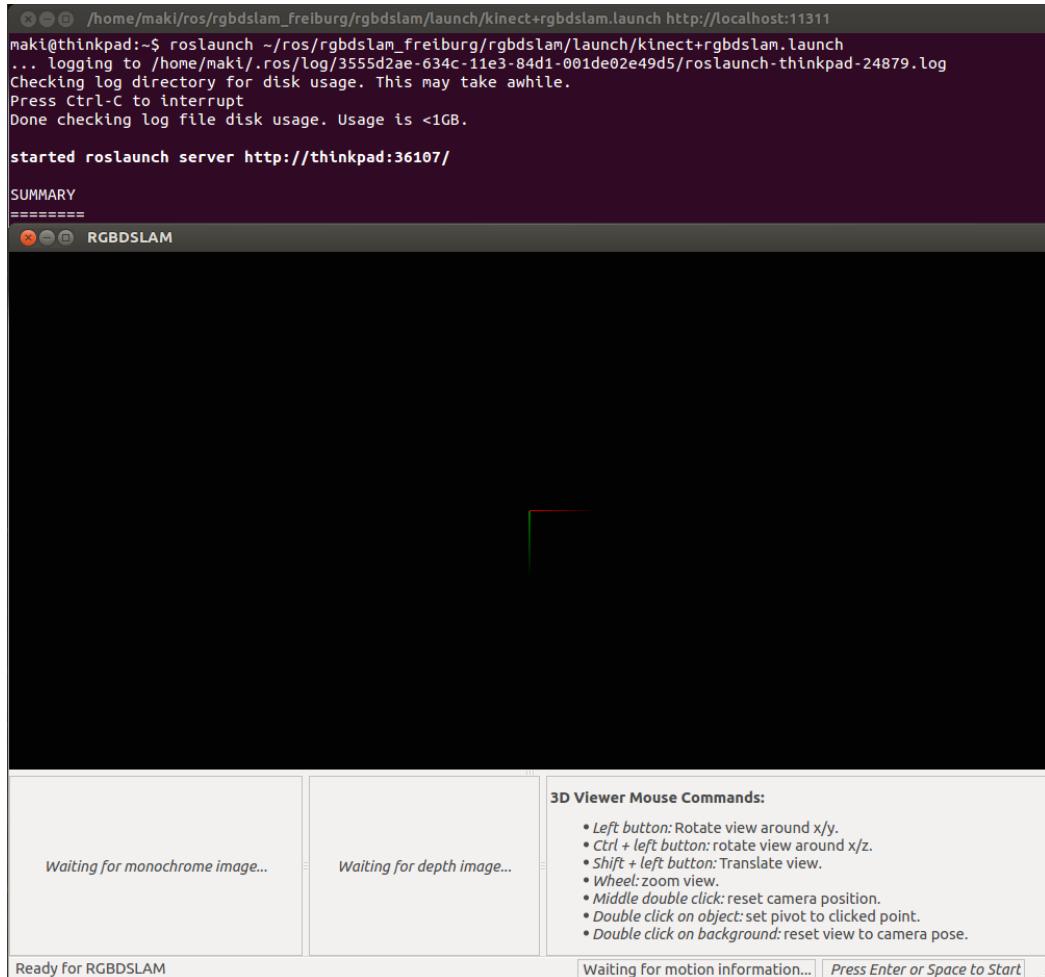
Grafikon 3.1.: Shematski pregled<sup>1</sup> RGBDSlam programa

Kao što je vidljivo iz grafikona 3.1. program je podijeljen u četiri osnovna dijela. Prvi dio računa značajke iz ulaznih slika u boji. RGBDSlam se oslanja na OpenCV [2] biblioteku u kojoj su implementirani SURF [1], SIFT [11] i ORB [14] algoritmi za pronađak značajki. Zatim se te značajke sparaju sa značajkama iz prethodnih slika. Drugi dio ispituje dubinu slika na lokacijama izračunatih značajki. Time je dobiveno znanje o 3D korespondencijama točaka između bilo koje dvije sličice. Zatim je na temelju tih korespondencija estimirana relativna transformacija između sličica upotrebom RANSACa. RANSAC Random Sample Consensus [5] je iterativna metoda estimiranja parametara matematičkog modela iz promatranih mjerena koji sadržavaju šum. Kako parovi estimiranih pozicija između sličica nisu globalno konzistentni treći dio programa optimizira graf pozicija upotrebom g<sup>2</sup>o algoritma. g<sup>2</sup>o General Framework for Graph Optimization [9] je okvir otvorenog koda za optimiziranje graph-based? nelinearnih error? funkcija. Algoritam u ovoj fazi daje globalno konzistentni 3D model promatrane okoline predstavljen oblakom točaka u boji. Takav oblak točaka je upotrebljen u diplomskom radu za stvaranje mreže trokuta. Zadnji četvrti dio upotrijebjava Octomap biblioteku kako bi generirao volumetrijski prikaz okoline. OctoMap [6] je efikasni probabilistički okvir za 3D mapiranje baziran na octree strukturi. Taj dio se uključuje posebnom datotekom prilikom pokretanja programa i nije korišten u ovom radu.

<sup>1</sup>Shema je preuzeta iz znanstvenog rada “An Evaluation of the RGB-D SLAM System” autora Endres, Hess, Burgard, Engelhard, Cremers, Sturm [4].

### 3.1.2. Pokretanje RGBDSLam programa

Instaliran program pokreće se preko komandne linije koristeći `roslaunch` koji je dio ROS [13] biblioteke i alata koji su detaljnije objašnjeni u poglavlju 2.2. Kao što je prikazano na slici 3.1. `roslaunch` za parametar prima XML datoteku s ekstenzijom `.launch` u kojoj su definirani parametri s kojima se pokreće program.



Slika 3.1.: Prikaz pokretanja RGBDSLam programa iz komandne linije

Prilikom upotrebe RGBDSLama nije bilo potrebe za mijenjanjem zadanih postavki te je korištena zadana `kinect+rgbdslam.launch` datoteka za pokretanje. Program podržava dva načina rada, automatski i ručni. Kod automatskog načina program neprestano uzima slike s kamere i procesira ih, što u kratkom vremenu rezultira velikom količinom podataka. Ručni način korisniku omogućava uzimanje slike na pritisak tipke Enter.

### **3.1.3. Snimanje scena RGBDSlam programom**

Za snimanje scena upotrijebljen je ručni način rada RGBDSlam programa. Prednost ručnog načina rada je što snimatelj kontrolira broj slika uzetih s kamere. Nedostatak je što je nezgodno jednoj osobi baratati s kamerom, gledati u računalo i pritiskati Enter za slikanje. Zato je pri snimanju scena sudjelovalo više osoba ili je korištena skripta<sup>1</sup> koja umjesto korisnika šalje signal Enter programu nakon proizvoljnog broja sekundi. Tijekom snimanja trebalo je obratiti pozornost na značajke scene koja se snima kako bih program mogao spariti značajke s prethodnom scenom. Tijekom izrade diplomskog rada snimljeno je osam scena odnosno prostorija koje su obrađene u poglavljju 4. Snimljene scene su spremane u .pcd formatu i kao takve korištene za daljnju obradu u diplomskom radu.

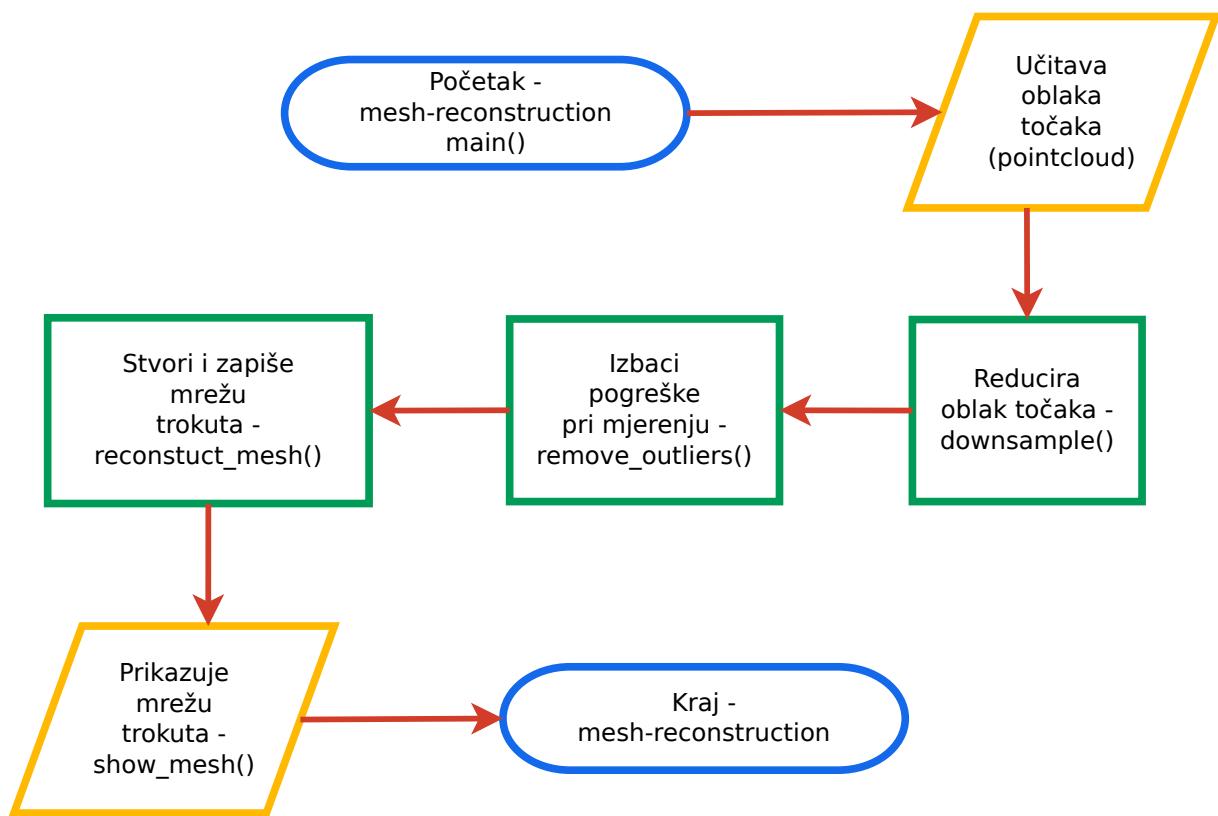
---

<sup>1</sup>Skripta za slikanje je dostupna u prilogu.

### 3.2. Izgradnja 3D modela scene pomoću mreže trokuta

Izgradnja 3D modela scene pomoću mreže trokuta je implementirana u programu nazvanom `mesh-reconstruction`.<sup>1</sup> Program se intenzivno oslanja na biblioteku PointCloud koja je opisana u podoglavlju 2.3. Kao što je vidljivo iz grafikona 3.2. program je podijeljen u pet osnovnih funkcija:

- Učitavanje oblaka točaka snimljenih RGBDSlam programom.
- Reduciranje oblaka točaka.
- Uklanjanje pogrešaka pri mjerenu.
- Stvaranje i zapisivanje mreže trokuta.
- Prikaz mreže trokuta.



Grafikon 3.2.: Dijagram toka programa `mesh-reconstruction`

U sljedećim podoglavlјima dan je pregled funkcija i PCL [15] klase na kojima se baziraju. Također na slici 3.2. se vidi kako izgleda pokretanje programa, što sve ispisuje na standardni izlaz te kako prikazuje mrežu trokuta.

<sup>1</sup>Program `mesh-reconstruction` je slobodan program dostupan pod uvjetima MIT licence. Izvorni kod se nalazi u prilogu te na web stranici [github.com/msvalina/](https://github.com/msvalina/)

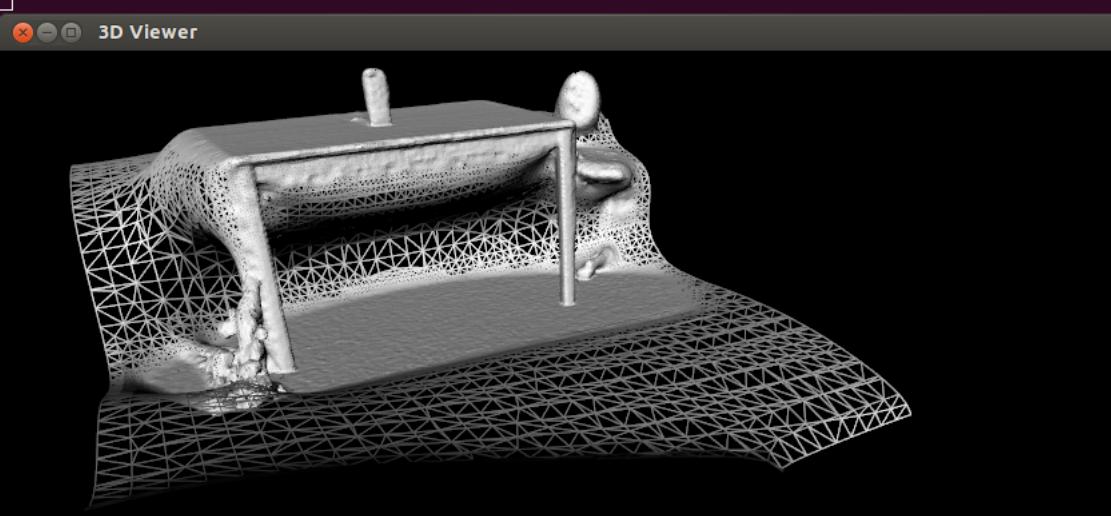
```
ranger:source/masters-thesis/mesh-reconstruction-build-desktop-Qt_4_8_1_in_PATH_System_Debug
maki@thinkpad:./mesh-reconstruction table_scene_lms400.pcd
Started - downsample() with VoxelGrid
PointCloud before filtering: 460400 data points (x y z intensity distance sid).
PointCloud after filtering: 41049 data points (x y z intensity distance sid).
Finished - downsample() with VoxelGrid

Started - remove_outliers() with StatisticalOutlierRemoval
PointCloud before filtering: 41049 data points (x y z intensity distance sid).
PointCloud after filtering: 39488 data points (x y z intensity distance sid).
Finished - remove_outliers() with StatisticalOutlierRemoval

Started - reconstruct_mesh() with Poisson
Failed to find match for field 'rgb'.
PointCloud loaded: 39488 data points
Finshed - reconstruct_mesh() with Poisson

Started - show_mesh() with PCLVisualizer

```

A screenshot of a 3D visualization application window titled "3D Viewer". The window displays a 3D point cloud or mesh reconstruction of a table scene. The table is a rectangular structure with a central support leg and two smaller legs at the ends. A small white cylindrical object is placed on top of the table. The base of the table and the floor are also visible as a triangular mesh. The overall color palette is grayscale, with some darker areas representing shadows or depth.

Slika 3.2.: Prikaz pokretanja programa `mesh-reconstruction` iz terminala

### 3.2.1. Pregled main() funkcije

Ispis koda 3.1.: Izvorni kod main() funkcije

```
1 int main (int argc, char *argv[])
{
3     downsample (argc, argv);
4     remove_outliers (argc, argv);
5     pcl::PolygonMesh mesh_of_triangles;
6     reconstruct_mesh (argc, argv, mesh_of_triangles);
7     show_mesh (mesh_of_triangles);
8     return 0;
9 }
```

Kao što se vidi iz ispisa koda 3.1. ideja je da funkcija bude što manja te da se iz nje samo pozivaju druge funkcije.

### 3.2.2. Učitavanje oblaka točaka

Program se sastoji od funkcija downsample, remove\_outliers, reconstruct\_mesh i show\_mesh. Na početku svake funkcije program učitava oblak točaka, obrađuje ga i zapisuje na izlazu iz funkcije kako bi prije i poslije svake operacije bio dostupan. Za to koristi PCDReader i PCDWriter klase. Predložak takvog koda se nalazi u ispisu koda 3.2. Nakon učitavanja oblaka točaka **reader** objektom, nad njim se vrše operacije npr. reduciranje oblaka točaka. Nakon toga kreiranjem i korištenjem **writer** objekta promjeni oblak se zapisuje u datoteku.

Ispis koda 3.2.: Predložak izvornog koda za učitavanje oblaka točaka

```
1 // Init cloud variables
2 pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2());
3 pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2());
4
5 // Fill in the cloud data
6 pcl::PCDReader reader;
7 reader.read ("pointcloud.pcd", *cloud);
8 /*
9  * Do something with cloud e.g. downsample pointcloud
10 */
11 // Write cloud to a file
12 pcl::PCDWriter writer;
13 writer.write ("pointcloud-downsampled.pcd",
14             *cloud_filtered, Eigen::Vector4f::Zero(),
15             Eigen::Quaternionf::Identity(), false);
```

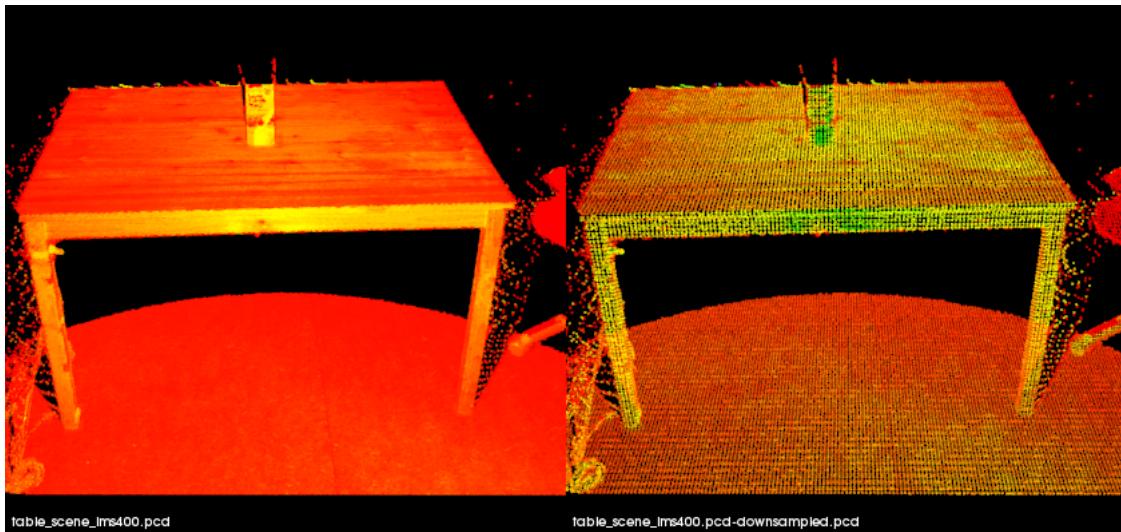
### 3.2.3. Reduciranje oblaka točaka

Reduciranje obalaka ne unosi bitne gubitake informacija, a izvodi se zbog lakše obrade oblaka. Izvodi se pomoću `VoxelGrid` klase i implementirano je u `downsample()` funkciji. Dijelovi funkcije prikazani su u ispisu koda 3.3. `VoxelGrid` dolazi od riječi *volume pixel grid* i predstavlja niz malih kocaka u prostoru.

Ispis koda 3.3.: Dio izvornog koda za reduciranje točaka iz funkcije `downsample()`

```
1 // Create the filtering object  
2 pcl::VoxelGrid<pcl::PCLPointCloud2> vg;  
3 vg.setInputCloud (cloud);  
4 // voxel size to be 1cm^3  
5 vg.setLeafSize (0.01f, 0.01f, 0.01f);  
vg.filter (*cloud_filtered);
```

Kao što se vidi iz ispisa koda 3.3. nakon kreiranja objekta `vg` predaje mu se oblak točaka nad kojim se vrši reduciranje. Postavlja se veličina kocke (*voxel*) u našem slučaju to je  $1\text{cm}^3$ . Nad tim oblakom prilikom filtriranja će se kreirati mreža kocaka te će se sve točke unutar jedne kocke zamjeniti centralnom točkom. Tim postupkom značajno se smanjuje broj točaka u oblaku kao što je vidljivo iz slike 3.3.



Slika 3.3.: Oblak točaka `table_scene`<sup>1</sup> lijevo prije `downsample()` 460400 točaka i poslije desno 41049 točaka

<sup>1</sup>Oblak točaka `table_scene_lms400.pcd` je objavljen pod uvjetima BSD licence izvor: [github.com/PointCloudLibrary/.../table\\_scene\\_lms400.pcd](https://github.com/PointCloudLibrary/.../table_scene_lms400.pcd)

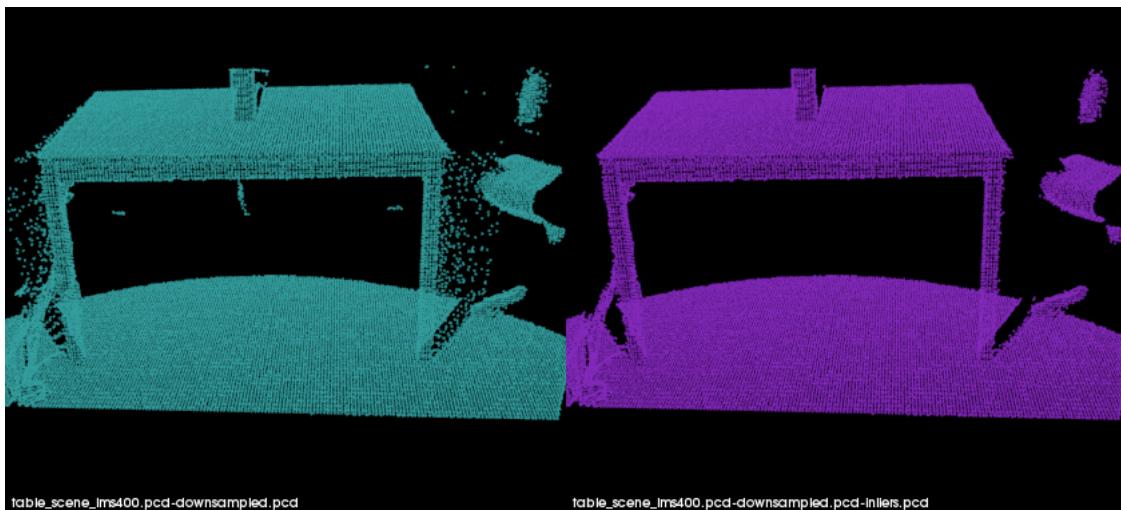
### 3.2.4. Uklanjanje pogrešaka pri mjerenuju

Pogreške pri mjerenuju su sastavni dio svakog mjernog uređaja. U slučaju Kineckt senzora često se javljaju odudarajuće vrijednosti (engl. *outliers*). PointCloud biblioteka ima ugrađenu `StatisticalOutlierRemoval` klasu koja uklanja odudarajuće vrijednosti, a implementirana je u funkciji `remove_outliers()`. Iz ispisa koda 3.4. se vidi kako se klasa koristi.

Ispis koda 3.4.: Dio izvornog koda za uklanjanje odudarajućih vrijednosti iz funkcije `remove_outliers()`

```
1 // Create the filtering object
2 pcl::StatisticalOutlierRemoval<pcl::PCLPointCloud2> sor;
3 sor.setInputCloud (cloud);
4 // Set number of neighbors to analyze
5 sor.setMeanK (50);
6 sor.setStddevMulThresh (1.0);
7 sor.filter (*cloud_filtered);
```

Nakon kreiranja objekta `sor` i predavanja oblaka postavljena su još dva parametra. Prvi `setMeanK` je broj susjednih točaka koje će filter analizirati. Drugi `setStddevMulThresh` postavlja multiplikator praga standardne devijacije. Algoritam dva puta prolazi kroz oblak točaka. U prvoj iteraciji računa srednju vrijednost udaljenosti svake točke do njenih  $K$  susjeda. Zatim računa očekivanje  $\mu$  i standardnu devijaciju  $\sigma$  svih udaljenosti kako bi mogao odrediti prag udaljenosti. Prag udaljenosti je određen jednadžbom  $\mu + \text{StddevMulThresh} \cdot \sigma$ . U sljedećoj iteraciji točke će biti označene kao outlier ukoliko su njihove srednje vrijednosti udaljenosti od susjeda veće od praga. Rezultati rada funkcije se vide na slici 3.4.



Slika 3.4.: Oblak točaka lijevo poslije `downsample()` 41049 točka i desno poslije `remove_outliers()` 39488 točka

### 3.2.5. Stvaranje i zapisivanje mreže trokuta

Nakon pripreme oblaka točaka funkcijama `downsample()` i `remove_outliers()` slijedi stvaranje mreže trokuta unutar funkcije `mesh_reconstruction()`. Stvaranje mreže trokuta se može podijeliti u tri koraka. Prvi je estimiranje normala nad oblakom točaka. Drugi je spajanje estimiranih normala i oblaka točaka u zajedniči oblak točaka s normalama. Treći korak je pozivanje algoritma za stvaranje mreže nad novo stvorenim oblakom.

Ispis koda 3.5.: Dio izvornog koda za estimaciju normala iz funkcije `reconstruct_mesh()`

```
1 // Normal estimation
2 pcl::NormalEstimation<PointType, Normal> normEst;
3 pcl::PointCloud<Normal>::Ptr normals (new pcl::PointCloud<Normal>());
4
5 // Create kd-tree representation of cloud,
6 // and pass it to the normal estimation object.
7 pcl::search::KdTree<PointType>::Ptr tree (new
8     pcl::search::KdTree<PointType>());
9 tree->setInputCloud (cloud);
10 normEst.setInputCloud (cloud);
11 normEst.setSearchMethod (tree);
12 // Use 20 neighbor points for estimating normal
13 normEst.setKSearch (20);
14 normEst.compute (*normals);
```

Iz ispisa koda 3.5. se vidi da je prije estimiranja normala nad oblakom točaka potrebno inicijalizirati objekt za spremanje normala i za estimaciju. Nakon toga definira se stablo za pretraživanje oblaka tipa `KdTree`.<sup>1</sup> Stablu se tada predaje oblak za pretraživanje. Objektu za estimaciju `normEst` tada se predaje oblak i stablo te broj susjednih točaka nad kojima se vrši estimacija normala<sup>2</sup>.

Nakon estimacije normala slijedi spajanje estimiranih normala i oblaka u novi oblak točaka s normalama. Kao što je prikazano u ispisu koda 3.6. Taj oblak točaka je prikazan na slici 3.5.

---

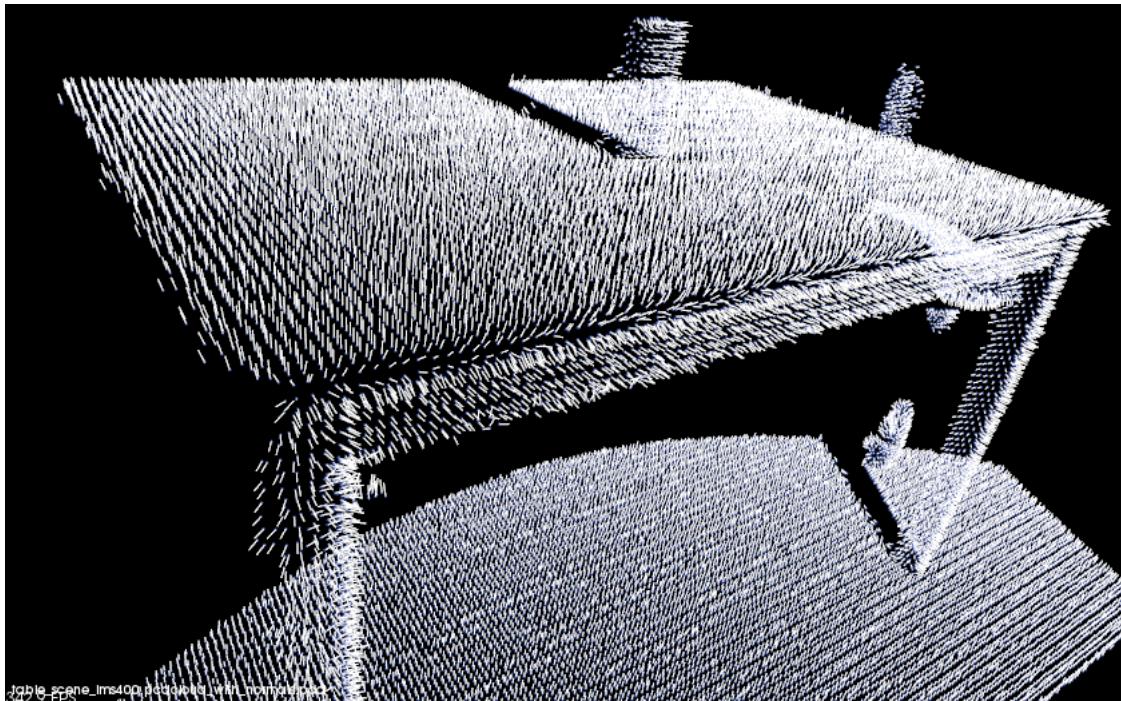
<sup>1</sup>K dimenzionalno stablo [12] je detaljno objašnjeno i na stranici [http://pointclouds.org/documentation/tutorials/kdtree\\_search.php](http://pointclouds.org/documentation/tutorials/kdtree_search.php)

<sup>2</sup>Estimacija normala detaljno je objašnjena na stranici [http://pointclouds.org/documentation/tutorials/normal\\_estimation.php](http://pointclouds.org/documentation/tutorials/normal_estimation.php)

Ispis koda 3.6.: Dio izvornog koda za stvaranju mreže iz funkcije `reconstruct_mesh()`

```
// Concatenate the XYZ and normal fields
2  pcl::PointCloud<PointTypeN>::Ptr cloud_with_normals (new
3      pcl::PointCloud<PointTypeN>);
4  pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);
5  // cloud_with_normals = cloud + normals
6
7  // Create search tree
8  pcl::search::KdTree<PointTypeN>::Ptr tree2 (new
9      pcl::search::KdTree<PointTypeN>);
10 tree2->setInputCloud (cloud_with_normals);
11
12 // Initialize objects
13 // psn - for surface reconstruction algorithm
14 // triangles - for storage of reconstructed triangles
15 pcl::Poisson<PointTypeN> psn;
16 pcl::PolygonMesh triangles;
17
18 psn.setInputCloud(cloud_with_normals);
19 psn.setSearchMethod(tree2);
20 psn.reconstruct (triangles);
21 psn.setOutputPolygons(false);
```

Nad stvorenim oblakom s normalama stvara se stablo za pretraživanje. Zatim se inicijaliziraju objekti `psn` i `triangles`. `psn` predstavlja Poisson<sup>1</sup> algoritam za stvaranje mreže trokuta. `triangles` je objekt tipa `PolygonMesh` za spremanje izračunatih koordinata trokuta. Algoritmu se sada predaje ulazni oblak, stablo pretraživanja i poziva se rekonstrukcija.



Slika 3.5.: Prikaz oblaka točaka `tablescene` s estimiranim normalama

<sup>1</sup>Poisson algoritam su razvili Michael Kazhdan i Matthew Bolitho, objavljen je pod BSD licencom. Izvor: <http://www.cs.jhu.edu/~misha/Code/PoissonRecon/Version5.5/>

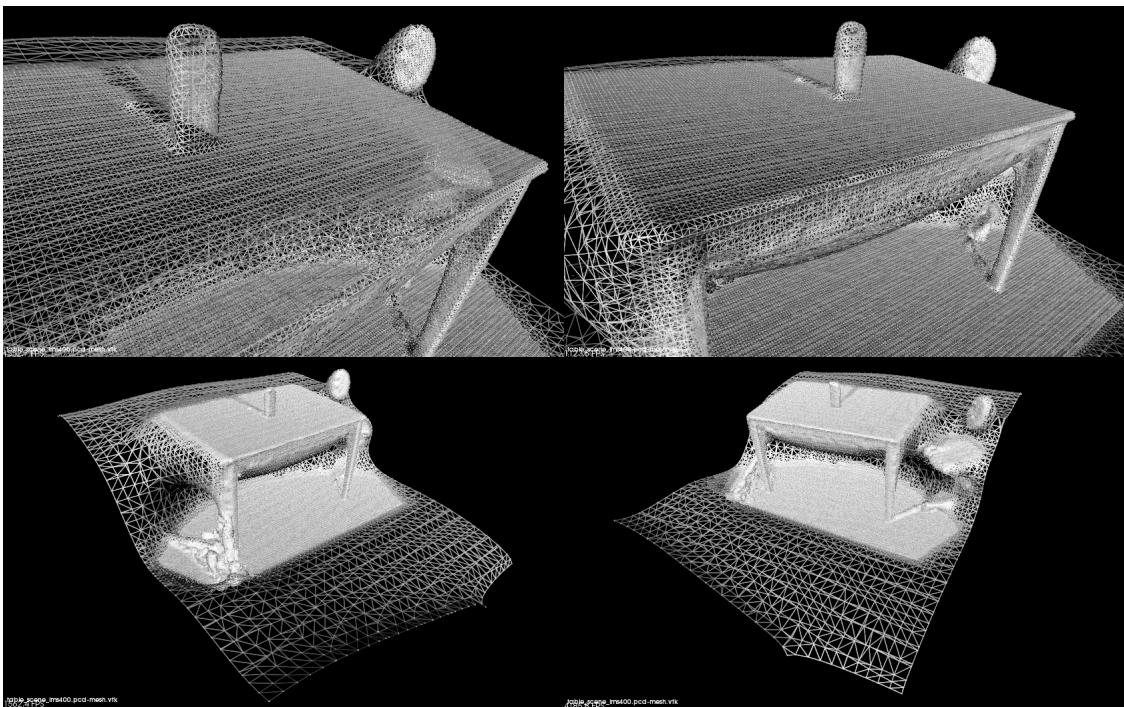
Ispis koda 3.7. prikazuje korištenje klase `saveVTKFile` za spremanje objekta `triangles` u datoteku s vtk ekstenzijom.

Ispis koda 3.7.: Dio izvornog koda za zapisivanju mreže iz funkcije `reconstruct_mesh()`

```
1 // Write reconstructed mesh
2 if (argc < 2){
3     pcl::io::saveVTKFile
4         ("pointcloud-downsampled-outliers-mesh.vtk",
5          triangles);
6 }
7 else {
8     std::string str;
9     str.append(argv[1]).append("-mesh.vtk");
10    pcl::io::saveVTKFile (str, triangles);
11 }
```

### 3.2.6. Prikazivanje mreže trokuta

Prikazivanje mreže trokuta omogućava `PCLVisualizer` klasu. Ista klasa se koristi u komandno linijskom programu za prikaza oblaka točaka `pcl_vieweru`. U ispisu koda 3.8. se vidi jednostavnost upotrebe klase. Nakon kreiranja objekta `viewer` i postavljanja parametara poziva se beskonačna petlja. Metoda `spinOnce()` izvodi crtanje mreže, prikazivanje na ekranu i daje `vieweru` vremena za procesiranje i time omogućava interaktivnost s mrežom. Klikom na tipku `q` izlazi se iz petlje i program završava. Slika 3.6. prikazuje izgled mreže iz četiri pogleda.



Slika 3.6.: Prikaz mreže trokuta funkcijom `show_mesh()`

Ispis koda 3.8.: Izvorni kod funkcije `show_mesh()`

```
1 void show_mesh (const pcl::PolygonMesh& mesh_of_triangles)
{
3     std::cout << "Started - show_mesh() with PCLVisualizer\n";
5     // Create viewer object and show mesh
6     boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
7         pcl::visualization::PCLVisualizer ("3D Viewer"));
8     viewer->setBackgroundColor (0, 0, 0);
9     viewer->addPolygonMesh (mesh_of_triangles, "sample mesh");
10    viewer->initCameraParameters ();
11    while (!viewer->wasStopped ())
12    {
13        viewer->spinOnce (100);
14        boost::this_thread::sleep
15            (boost::posix_time::microseconds (100000));
16    }
17 }
```

## **4. REZULTATI**

Ovdje će prikazati snimke i statističke podatke o njima. Trebam smisliti kako ispitati kvalitetu dobivenih rezulatata. Problem je što nemam s čime usporediti. Plus ja ne bih trebao ispitivati kvalitetu dobivenih rezultata već funkcionalnost postupka izrađivanja 3d modela scene pomoću 3d kamere.

## **5. ZAKLJUČAK**

## LITERATURA

- [1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.
- [2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [3] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.
- [4] Felix Endres, Juergen Hess, Nikolas Engelhard, Juergen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, Minnesota, 2012.
- [5] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [6] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at <http://octomap.github.com>.
- [7] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 61–70, Aire-la-Ville, Switzerland, 2006. Eurographics Association.
- [8] Michael Kazhdan, Allison Klein, Ketan Dalal, and Hugues Hoppe. Unconstrained iso-surface extraction on arbitrary octrees. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 125–133, Aire-la-Ville, Switzerland, 2007. Eurographics Association.
- [9] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.
- [10] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *COMPUTER GRAPHICS*, 21(4):163–169, 1987.
- [11] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- [12] Andrew Moore. A tutorial on kd-trees. Technical report, 1991.
- [13] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [14] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *International Conference on Computer Vision*, Barcelona, 11/2011 2011.

- [15] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011 2011.

# SAŽETAK

# ŽIVOTOPIS

## PRILOZI

Prevodenje i instalacija RGBDSlam programa