SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

IZGRADNJA 3D MODELA SCENE POMOĆU 3D KAMERE

Diplomski rad

Marijan Svalina

Osijek, 2013.

Sadržaj

1. Uvod **1**
1.1. Zadatak diplomskog rada 1

2. Pregled korištenih tehnologija i algoritama **2**
2.1. Microsof Kineckt 3D kamera 2
2.2. Biblioteka Pointcloud 2
2.3. Istovremena lokalizacija i mapiranje 2
2.4. Poisson algoritam za rekonstrukciju površine 2

3. Izgradnja 3D modela scene **3**
3.1. Snimanje scene 3D kamerom i RGBDSlam programom 3
3.2. Izgradnja 3D modela scene pomoću mreže trokuta 4

4. Rezultati **12**

5. Zaključak **13**

Literatura **14**

Sažetak **15**

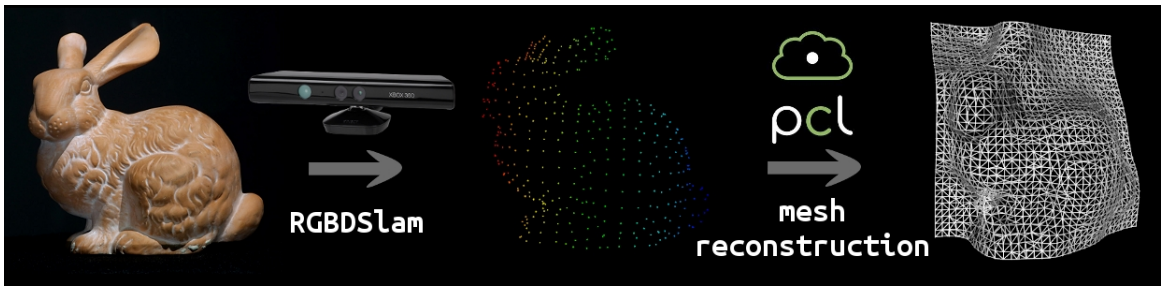
Životopis **16**

Prilozi **17**

1. UVOD

1.1. Zadatak diplomskog rada

PRIVREMEN C/P opisa diplomskog
Program RGBDSLAM raspoloživ u okviru programske biblioteke 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 opisa projekta

2. PREGLED KORIŠTENIH TEHNOLOGIJA I ALGORITAMA

2.1. Microsof Kineckt 3D kamera

2.2. Biblioteka Pointcloud

2.3. Istovremena lokalizacija i mapiranje

2.4. Poisson algoritam za rekonstrukciju površine

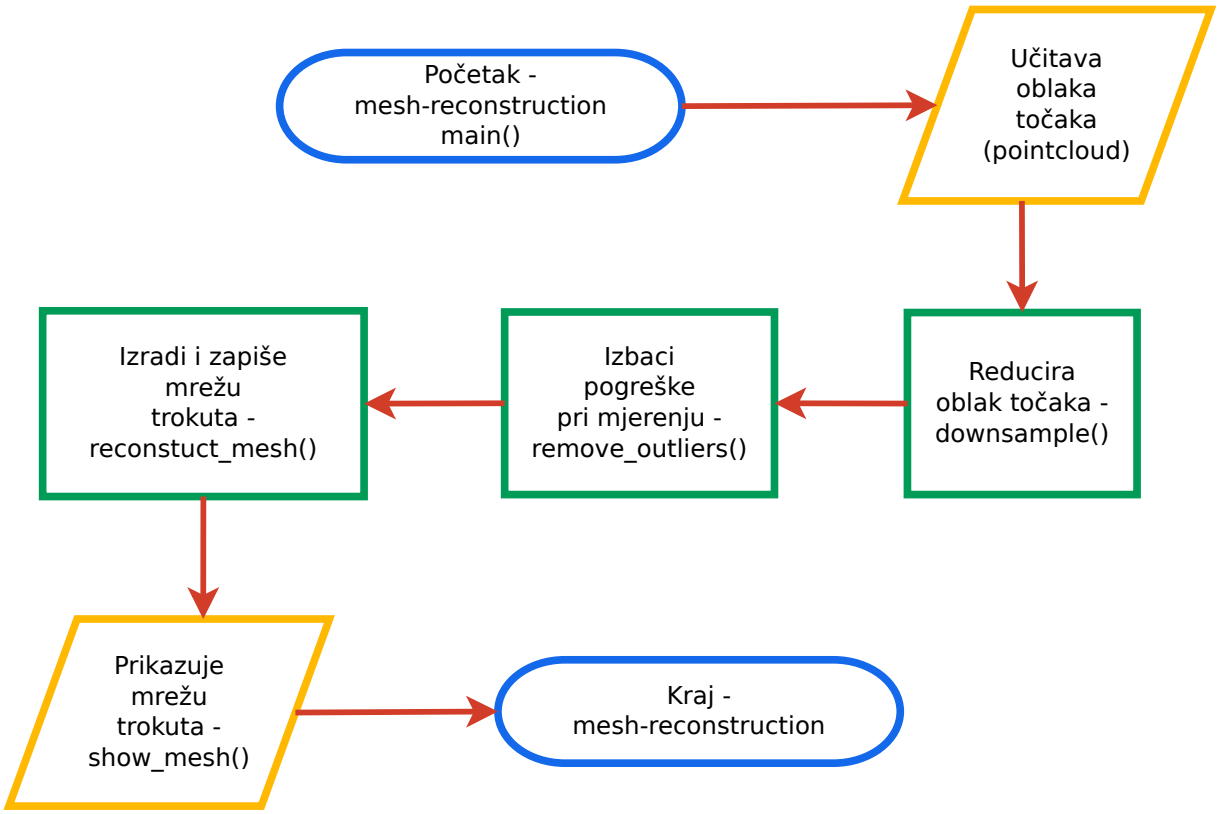
3. IZGRADNJA 3D MODELA SCENE

3.1. Snimanje scene 3D kamerom i RGBDSlam programom

3.2. Izgradnja 3D modela scene pomoću mreže trokuta

Izgradnja 3D modela scene pomoću mreže trokuta je implementirana u programu nazvanom `mesh-reconstruction`.[1] Program se intenzivno oslanja na biblioteku PointCloud koja je opisana u potpoglavlju 2.2. Kao što je vidljivo iz grafikona 3.1. program je podijeljen u pet osnovnih funkcija:

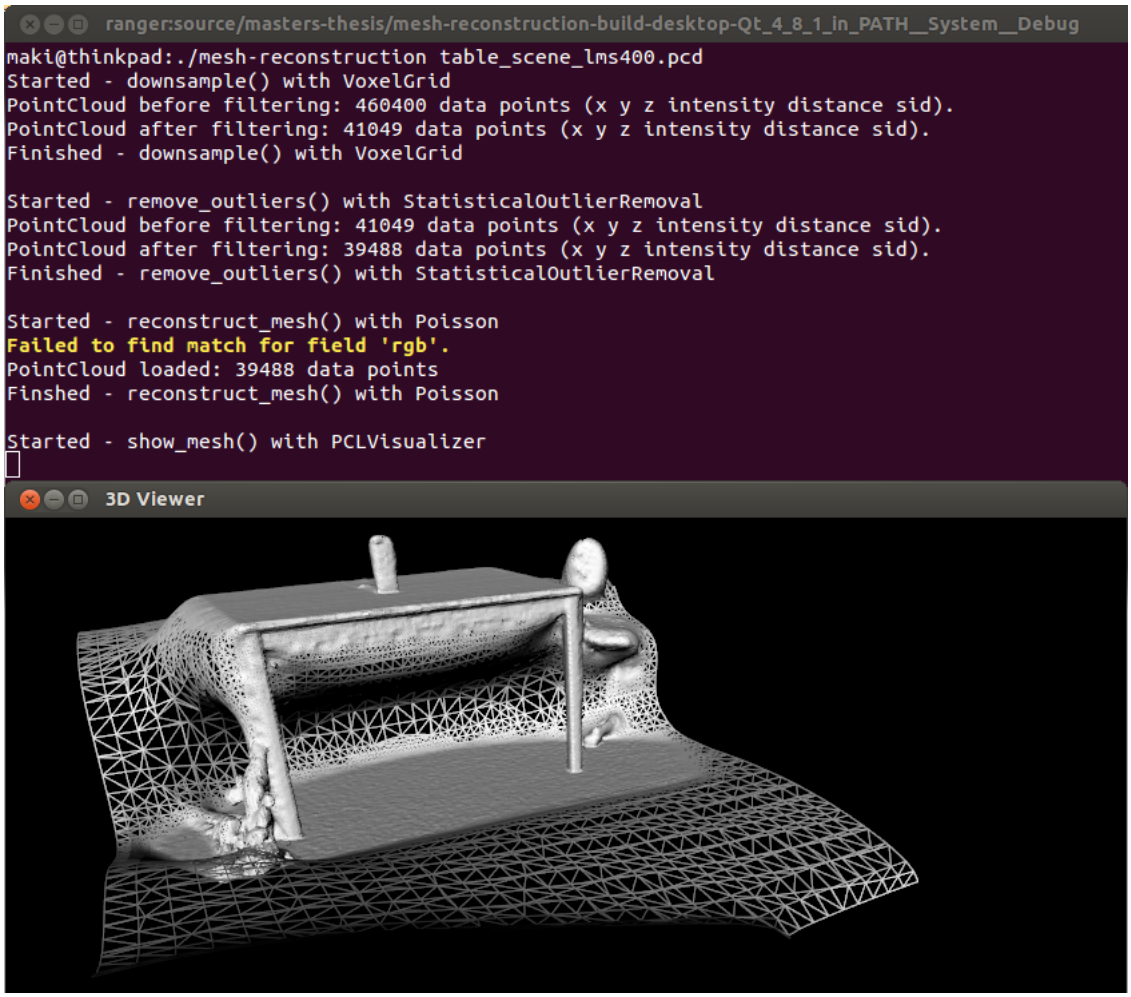
- Učitavanje oblaka točaka snimljenih RGBDSlam programom.
- Reduciranje oblaka točaka.
- Uklanjanje pogrešaka pri mjerenju.
- Izrađivanje i zapisivanje mreže trokuta.
- Prikaz mreže trokuta.



Grafikon 3.1.: Dijagram toka programa `mesh-reconstruction`

U sljedećim potpoglavljima dan je pregled funkcija i PCL klasa nad kojim se baziraju. Također na slici 3.1. se vidi kako izgleda pokretanje programa, što sve ispisuje na standardni izlaz te kako prikazuje mrežu trokuta.

[1] Program `mesh-reconstruction` je slobodan program dostupan pod uvijetima MIT licence. Izvorni kod se nalazi na DVD-u te na web stranici github.com/msvalina/



Slika 3.1.: 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);
      remove_outliers (argc, argv);
5     pcl::PolygonMesh mesh_of_triangles;
      reconstruct_mesh (argc, argv, mesh_of_triangles);
7     show_mesh (mesh_of_triangles);
      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 učitava podatke na početku svake funkcije, te ih zapisuje na izlazu iz funkcije kako bih prije i poslije svake operacije bio dostupan oblak točaka. Za to koristi `PCDReader` i `PCDWriter` klase. Primjer takvog koda se nalazi u ispisu koda 3.2.

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

```
    // Init cloud variables
    pcl::PCLPointCloud2::Ptr cloud (new pcl::PCLPointCloud2());
    pcl::PCLPointCloud2::Ptr cloud_filtered (new pcl::PCLPointCloud2());

    // Fill in the cloud data
    pcl::PCDReader reader;
    reader.read ("pointcloud.pcd", *cloud);
    /*
     * Do something with cloud
     */
    // Write cloud to a file
    pcl::PCDWriter writer;
    writer.write ("pointcloud-downsampled.pcd",
            *cloud_filtered, Eigen::Vector4f::Zero(),
            Eigen::Quaternionf::Identity(), false);
```

3.2.3. Reduciranje oblaka točaka

Reduciranje obalaka ne unosi gubitak informacija, a izvodi se zbog lakše daljnje 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()`

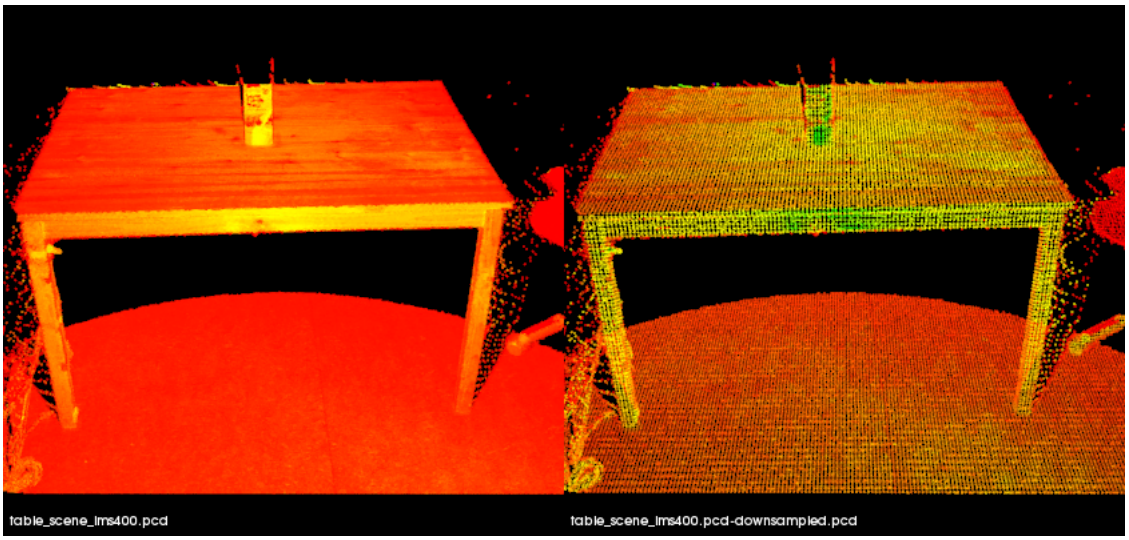
```
    // Create the filtering object
    pcl::VoxelGrid<pcl::PCLPointCloud2> vg;
    vg.setInputCloud (cloud);
    // voxel size to be 1cm^3
    vg.setLeafSize (0.01f, 0.01f, 0.01f);
    vg.filter (*cloud_filtered);
```

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 3cm^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 3.2.

3.2.4. Uklanjanje pogrešaka pri mjerenju

Šum pri mjerenju je sastavni dio svakog mjernog uređaja pa tako i Kineckt kamere. PointCloud biblioteka ima ugrađenu `StatisticalOutlierRemoval` klasu koja uklanja šum te je implementirana u funkciji `remove_outlieres()`. Iz ispisa koda 3.4. se vidi kako se klasa koristi.

[2] Oblak točaka `table_scene_lms400.pcd` je objavljen pod uvijetima BSD licence - izvor

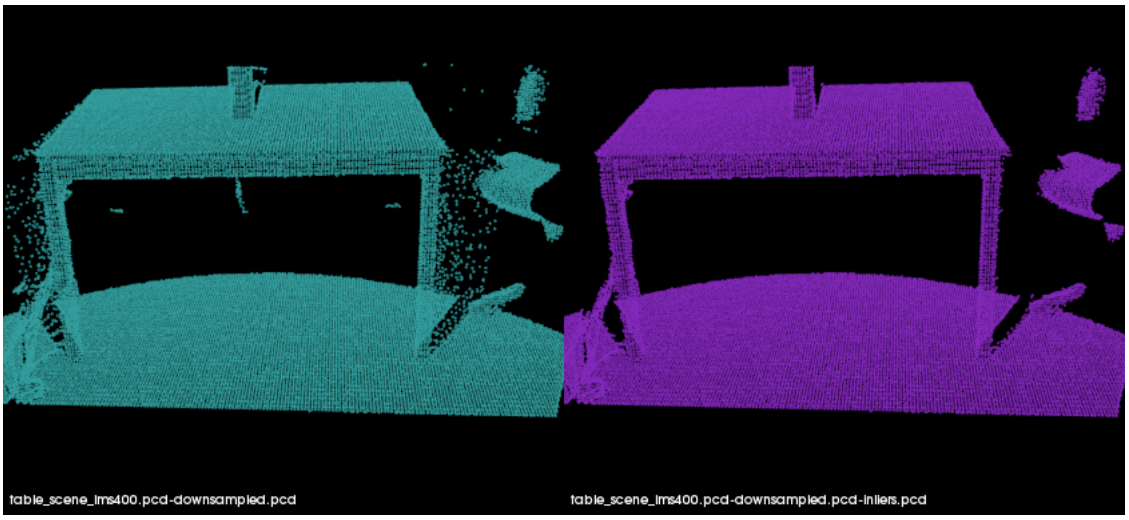


Slika 3.2.: Oblak točaka *table_ scene*[2] prije i poslije `downsample()` funkcije

Ispis koda 3.4.: Dio izvornog koda iz funkcije `remove_outliers()`

```
    // Create the filtering object
2   pcl::StatisticalOutlierRemoval<pcl::PCLPointCloud2> sor;
    sor.setInputCloud (cloud);
4   // Set number of neighbors to analyze
    sor.setMeanK (50);
6   sor.setStddevMulThresh (1.0);
    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 analizirat. Drugi `setStddevMulThresh` pak kaže da će sve točke u okolini ispitane točke čije su udaljenosti veće od jedne standardne devijacije očekivane udaljenosti biti označne kao šum (*outlier*) i odbačene. Rezultati rada funkcije se vide na slici 3.3.



Slika 3.3.: Oblak točaka: lijevo poslije `downsample()` i desno poslije `remove_outliers()`

3.2.5. Izrađivanje i zapisivanje mreže trokuta

Nakon pripreme oblaka točaka funkcijama `downsample()` i `remove_outliers()` slijedi izrađivanje mreže trokuta unutar funkcije `mesh_reconstruction()`. Izrađivanje 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 izrađivanje mreže nad novo stvorenim oblakom.

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

```
1     // Normal estimation
      pcl::NormalEstimation<PointType, Normal> normEst;
3     pcl::PointCloud<Normal>::Ptr normals (new pcl::PointCloud<Normal>);

5     // Create kdtree representation of cloud,
      // and pass it to the normal estimation object.
7     pcl::search::KdTree<PointType>::Ptr tree (new
              pcl::search::KdTree<PointType>);
9     tree->setInputCloud (cloud);
      normEst.setInputCloud (cloud);
11    normEst.setSearchMethod (tree);
      // Use 20 neighbor points for estimating normal
13    normEst.setKSearch (20);
      normEst.compute (*normals);
```

Iz ispisa koda 3.5. se vidi da je prije estimiranja normala nad oblakom točaka potrebno je inicijalizirati objekt za spremanje normala i za estimaciju. Nakon toga definira se stablo za pretraživanje oblaka tipa `KdTree`.[3] 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[4].

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 3.4.

[3] K dimenzionalno stablo je detaljno objašnjeno na stranici pointclouds.org/documentation

[4] Estimacija normala detaljno je objašnjena na stranici pointclouds.org/documentation

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

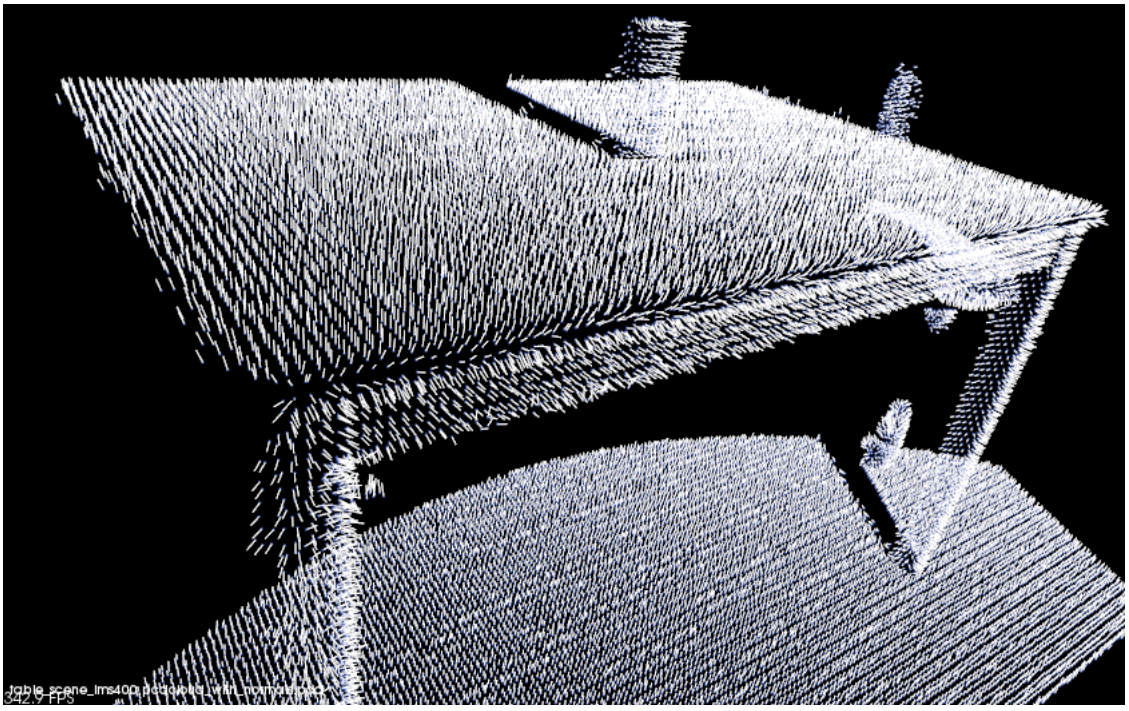
```
    // Concatenate the XYZ and normal fields
    pcl::PointCloud<PointTypeN>::Ptr cloud_with_normals (new
            pcl::PointCloud<PointTypeN>);
    pcl::concatenateFields (*cloud, *normals, *cloud_with_normals);
    // cloud_with_normals = cloud + normals

    // Create search tree
    pcl::search::KdTree<PointTypeN>::Ptr tree2 (new
            pcl::search::KdTree<PointTypeN>);
    tree2->setInputCloud (cloud_with_normals);

    // Initialize objects
    // psn - for surface reconstruction algorithm
    // triangles - for storage of reconstructed triangles
    pcl::Poisson<PointTypeN> psn;
    pcl::PolygonMesh triangles;

    psn.setInputCloud(cloud_with_normals);
    psn.setSearchMethod(tree2);
    psn.reconstruct (triangles);
    psn.setOutputPolygons(false);
```

Nad stvorenim oblakom s normalama stvara se stablo za pretraživanje. Zatim se inicijaliziraju objekti `psn` i `triangles`. `psn` predstavlja `Poisson`[5] algoritam za izrađivanje mreže trokuta. `triangles` je objekt tipa `PolygonMesh` za spremanje izražunatih koordinata trokuta. Algoritmu se sada predaje ulazni oblak, stablo pretraživanja i poziva se rekonstrukcija.



Slika 3.4.: Prikaz oblaka točaka `tablescene` s estimiranim normalama

[5]Poisson algoritam su razvili Michael Kazhdan i Matthew Bolitho, objavljen je pod BSD licencom. Službena stranica.

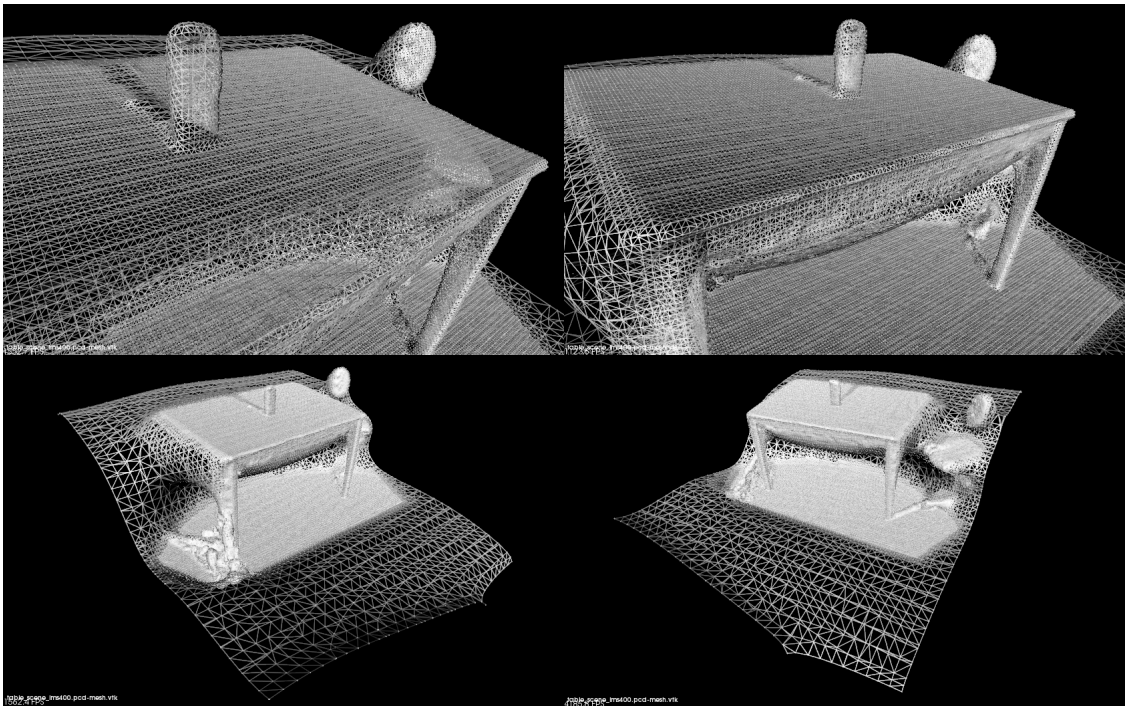
Ispis koda 3.7. prikazuje korištenje klase `saveVTKFile` za spremanje objekta `triangles` u datoteku s vtk ekstenzijom.

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

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

3.2.6. Prikazivanje mreže trokuta

Prikazivanje mreže trokuta omogućava `PCLVisualizer` klasa. Ista klasa se koristi u komandono 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 unutar koje se pokrene prozor s prikazom mreže trokuta. Klikom na tipku `q` izlazi se iz petlje i program završava. Slika 3.5. prikazuje izgled mreže iz četiri pogleda.



Slika 3.5.: Prikaz mreže trokuta funkcijom `show_mesh()`

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

```
void show_mesh (const pcl::PolygonMesh& mesh_of_triangles)
{
    std::cout << "Started - show_mesh() with PCLVisualizer\n";
    // Create viewer object and show mesh
    boost::shared_ptr<pcl::visualization::PCLVisualizer> viewer (new
          pcl::visualization::PCLVisualizer ("3D Viewer"));
    viewer->setBackgroundColor (0, 0, 0);
    viewer->addPolygonMesh (mesh_of_triangles, "sample mesh");
    viewer->initCameraParameters ();
    while (!viewer->wasStopped ())
    {
        viewer->spinOnce (100); boost::this_thread::sleep
            (boost::posix_time::microseconds (100000));
    }
    std::cout << "Finshed - show_mesh() with PCLVisualizer\n";
}
```

4. REZULTATI

5. ZAKLJUČAK

LITERATURA

[1] Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.

SAŽETAK

ŽIVOTOPIS

PRILOZI