6. Procesarea datelor RGB-D

Reprezentarea datelor RGB-D
Manipularea voxelilor

În acest laborator se vor analiza datele achiziţionate utilizându-se senzori cu lumină structurată. Datele furnizate de astfel de senzori sunt compuse din informaţie vizuală, reprezentată sub formă de imagini RGB, cât şi din distanţele cameră-suprafaţa vizualizată. Aceste distanţe sunt codate sub forma unui canal adiţional D. Procesarea informației vizuale 3D se va efectua cu ajutorul librăriei *Point Cloud Library* (PCL) `http://pointclouds.org/`. Instalarea ei se face în mod similar cu instalarea librăriei OpenCV, descrisă în aplicația 1 a acestui îndrumar.

6.1 Baze teoretice

Punctul, notat în continuare cu p, este cel mai simplu element care poate descrie o informaţie într-un spaţiul n-dimensional. Complexitatea acestuia este dată de proprietăţile ce îl definesc. O primă proprietate fundamentală a acestuia este legată de poziția sa *geometrică*. Considerând spaţiul Euclidian $\Re^3$, punctul p poate fi definit utilizându-se cele trei coordonate carteziene (x, y, z). În acest sens, punctul devine tridimensional. Cea de-a doua proprietate de bază este reprezintă de *culoare*. Asemenea pixelului utilizat la reprezentarea imaginilor, culoarea punctului p poate fi reprezentată prin cele trei culori fundamentale (Roşu, Verde şi Albastru)[1]. Mai multe astfel de puncte pot fi grupate pentru a se forma o densitate de puncte, care poate fi analizată ulterior. O astfel de aglomerare de puncte este denumită simbolic *nor de puncte*[2] şi va fi notată în continuare cu P. Spre exemplificare, în Fig. 6.1 este prezentat un nor de puncte ce descrie o scenă uzuală. Aceasta a fost percepută utilizându-se un senzor cu lumină structurată Microsoft Kinect®.

Norii de puncte conţin informaţia de bază achiziţionată din scena vizualizată. Coordonatele $[x_i, y_i, z_i]$ ale unui punct $p_i \in P$ sunt raportate faţă de un sistem de coordonate fix, având de obicei originea în centrul senzorului utilizat la achiziţie. Acest lucru simbolizează faptul că fiecare punct p_i reprezintă distanţa pe cele 3 axe de coordonate de la senzorul video la suprafaţa obiectului vizualizat. Unele dintre cele mai populare tehnici de măsurare a distanţelor 3D sunt:

- tehnici de *triangulaţie*, care estimează adâncimea (distanţa) prin identificarea punctelor corespondente în imaginile percepute de către doi senzori diferiţi, la acelaşi moment de timp (vezi aplicația 5 din acest îndrumar);
- sisteme cu *lumină structurată*, care estimează distanţa dintre obiect şi senzor prin proiectarea unui *şablon* luminos cu structură cunoscută. Relaţia liniară dintre grosimea liniilor proiectate şi distanta reală permite estimarea cu precizie a adâncimii obiectului, relativ la senzor. Datorită influenţelor luminii naturale (în spectrul infraroşu) acest tip de cameră poate fi utilizat doar în spaţii închise;

[1] Eng. *RGB - Red, Green, Blue*
[2] Eng. *Point Cloud*

(a)

(b)

Fig. 6.1 Exemple de nori de puncte 3D.

- senzorii *Time-of-Flight* (ToF), care măsoară distanţele 3D prin determinarea timpului de întârziere dintre transmiterea şi recepţionarea unui semnal. Cunoscându-se viteza de propagare a semnalului, distanţa d dintre senzor şi un punct 3D se poate calcula prin relaţiea:

$$d = \frac{c \cdot t}{2} \tag{6.1}$$

  unde c reprezintă viteza semnalului (e.g. viteza luminii în cazul senzorilor laser, sau cu infraroşu), iar t este timpul de întârziere dintre momentul de transmitere a semnalului şi recepţionarea lui.

Odată achiziţionat un nor de puncte 3D, utilizându-se una dintre metodele mai sus menţionate, acesta este stocat sub diferite forme impuse de aplicaţia în cauză. Un aspect important legat de reprezentarea norilor de puncte este acela că în aceste structuri se pot stoca multiple caracteristici ale punctelor. Spre exemplu, în cazul în care se utilizează un senzor de lumină structurată, pe lângă caracteristica geometrică (de distanţă) punctul 3D mai prezintă şi informaţie de culoare, normală la suprafaţă, grad de apartenenţă la un anumit grup de puncte sau distanţa Euclidiană faţă de originea senzorului video. Astfel, definiţia unui punct $p_i = [x_i, y_i, z_i]$ se poate generaliza după cum urmează:

$$p_i = [f_0, f_1, f_2, \ldots, f_n], \tag{6.2}$$

unde f_j reprezintă o caracteristică într-un anumit spaţiu de caracteristici, spre exemplu poziţie, culoare, clasa de apartenenţă, etc. Structura unui nor de puncte P poate fi stocată

sub formă matriceală:

$$P = \begin{bmatrix} x_1 & y_1 & z_1 & r_1 & g_1 & b_1 & n_{x_1} & n_{y_1} & n_{z_1} & e_1 & d_1 & \cdots \\ x_2 & y_2 & z_2 & r_2 & g_2 & b_2 & n_{x_2} & n_{y_2} & n_{z_2} & e_2 & d_2 & \cdots \\ & & & & & & & & & & & \cdots \\ x_{N_p} & y_{N_p} & z_{N_p} & r_{N_p} & g_{N_p} & b_{N_p} & n_{x_{N_p}} & n_{y_{N_p}} & n_{z_{N_p}} & e_{N_p} & d_{N_p} & \cdots \end{bmatrix}. \tag{6.3}$$

unde x_1, y_1, z_1 reprezintă coordonatele unui anumit punct 3D, r_1, g_1, b_1 culoarea asociată acestui punct, n este normala la suprafaţă, e_1 clasa obiectului faţă de care punctul aparţine, iar d_1 distanţa senzor video - punct 3D.

Pentru a se putea înţelege geometria scenei în jurul unui anumit punct de interes p_i, este necesară descoperirea setului de vecini P^k localizaţi în jurul punctului p_i. Unele dintre cele mai utilizate soluţii în acest scop este utilizarea tehnicilor de decompoziţie spatială în arbori ("KD-tree" sau "Octree") şi înlocuirea punctelor 3D prin cuburi de dimensiuni constante.

6.2 Cerinţe

1. Utilizându-se librăria PCL, să se încarce de pe HDD un nor de puncte RGB-D;
2. Să se elimine din nor punctele care nu conţin informaţie de adâncime;
3. Să se vizualizeze norul de puncte încărcat;
3. Să se identifice centrul de greutate al norului de puncte;
4. Să se translateze norul în originea sistemului de coordonate;
5. Să se vizualizeze norul de puncte translatat.

6.3 Codul sursă al aplicației

```
1  #include <stdio.h>
2  #include <iostream>
3  #include <pcl/point_types.h>
4  #include <pcl/point_cloud.h>
5  #include <pcl/io/pcd_io.h>
6  #include <pcl/visualization/cloud_viewer.h>
7  #include <pcl/common/common.h>
8  #include <pcl/common/transforms.h>
9  #include <pcl/filters/filter.h>
10
11 using namespace pcl;
12 using namespace std;
13
14 int main(int argc, char ** argv)
15 {
16     cout << "Laborator 6: Introducere in PCL" << endl;
17
18     string strCaleFisier;
19     PointXYZRGBA ptCentruDeGreutate;
20
21     // Crearea unei structuri de date nor de puncte
22     PointCloud <PointXYZRGBA>::Ptr nor_puncte (new PointCloud <↩
       PointXYZRGBA>);
23
```

```
24      if (argc != 2)
25      {
26          cout<<"Usage: aplicatie <cale-fisier-RGBD>"<<endl;
27          exit(0);
28      }
29      else
30      {
31          strCaleFisier = argv[1];
32      }
33
34      // Incarcarea unui nor de puncte 3D utilizand libraria PCL
35      if( io::loadPCDFile(strCaleFisier.c_str(), *nor_puncte))
36          cout << "Norul de puncte nu a putut fi citit." << endl;
37      else
38          cout << "Norul de puncte contine: " << nor_puncte->points.↩
               size() << " puncte 3D " << endl;
39
40      // Eliminarea punctelor cu valoare nula
41      vector <int> vIndex;
42      removeNaNFromPointCloud (*nor_puncte, *nor_puncte, vIndex);
43      cout<<"Norul de puncte filtrat contine: " << nor_puncte->points.↩
          size() << " puncte 3D " << endl;
44
45      // Vizualizarea norului de puncte 3D
46      visualization::CloudViewer viewer ("Viualizarea grafica a norului↩
            de puncte");
47      viewer.showCloud (nor_puncte);
48      while (!viewer.wasStopped ())
49      {}
50
51      // Determinarea centrului de greutate al scenei
52      float fSumX = 0.0f, fSumY = 0.0f, fSumZ = 0.0f;
53
54      for ( unsigned int a = 0 ; a < nor_puncte->points.size(); a ++)
55      {
56          fSumX += nor_puncte->points[a].x;
57          fSumY += nor_puncte->points[a].y;
58          fSumZ += nor_puncte->points[a].z;
59      }
60      cout<<"Coordonate centru de greutate (X, Y, Z): " <<
61              fSumX / nor_puncte->points.size() << ", " <<
62              fSumY / nor_puncte->points.size()<< " , "<<
63              fSumZ / nor_puncte->points.size() << endl;
64
65      // Salvarea coordonatelor centrului de greutate
66      ptCentruDeGreutate.x = fSumX / nor_puncte->points.size();
67      ptCentruDeGreutate.y = fSumY / nor_puncte->points.size();
68      ptCentruDeGreutate.z = fSumZ / nor_puncte->points.size();
69
70      // Translatoarea norului de puncte in coordonatele de origine ale↩
            scenei
71      PointXYZ ptTranslation;
```

```
72      ptTranslation.x = 0 - ptCentruDeGreutate.x;
73      ptTranslation.y = 0 - ptCentruDeGreutate.y;
74      ptTranslation.z = 0 - ptCentruDeGreutate.z;
75
76      for (unsigned int b = 0; b < nor_puncte->points.size(); b++)
77      {
78          nor_puncte->points[b].x += ptTranslation.x;
79          nor_puncte->points[b].y += ptTranslation.y;
80          nor_puncte->points[b].z += ptTranslation.z;
81      }
82
83      // Vizualizarea norului de puncte 3D
84      visualization::CloudViewer viewer_2 ("Viualizarea grafica a ↩
           norului de puncte");
85      viewer_2.showCloud (nor_puncte);
86      while (!viewer_2.wasStopped ())
87      {}
88
89      return 0;
90  }
```

6.4 Descrierea funcţiilor principale

```
35  io::loadPCDFile(const string& file_name, PointCloud<PointT> &cloud)
```

Încarcă un fişier tip nor de puncte.

- `file_name`: fişierul sursă;
- `cloud`: structura în care va fi încărcat norul de puncte.

```
42  removeNaNFromPointCloud(const PointCloud<PointT>& cloud_in, ↩
      PointCloud<PointT>& cloud_out, vector<int>& index)
```

- `cloud_in`: norul de puncte de intrare;
- `cloud_out`: norul de puncte filtrat;
- `index`: vectorul cu indecşii punctelor valide din norul de puncte de intrare.