Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования

«Московский политехнический университет»

Кафедра «СМАРТ-технологии»

**Лабораторная работа №2**

По дисциплине: «Разработка систем сбора и обработки данных»

По теме: «Обработка структурированных пространственных данных и отслеживание объектов»

Группа: 211-327

Выполнил студент:

Роднов К.А. \_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватели: Идиатуллов Т.Т.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2024

**Цель работы:** разработать алгоритм считывания и визуализации потоковых данных в виде динамически обновляемого облака точек многолучевого 3D-лидара

**Задачи:**

• Подготовить приложение считывания данных с 3D-лидара, сохраненных в файле потоковом формате и вычисления облака точек;

• Разработать функцию динамического обновления данных в заданной структуре (объекте) для хранения данных облака точек;

• Разработать функцию покадровой визуализации облака точек в изометрической проекции и в режиме «вид сверху» с заданным (настраиваемым) фреймрейтом (частотой);

• Разработать функцию фильтрации данных с использованием плоскостей отсечения (куб интереса);

• Разработать функцию детектирования объектов методом кластерного анализа;

• Разработать метод отслеживания «коридора проходимости» для прямолинейного движения;

**Ход работы:**

1) Реализовано отображение облака точек по средством считывания данных 3D-лидара, сохранённых в файле. Для отрисовки используется библиотека OpenGl. Присутствует возможность настроить скорость отрисовки и обновления данных перемещая ползунок.

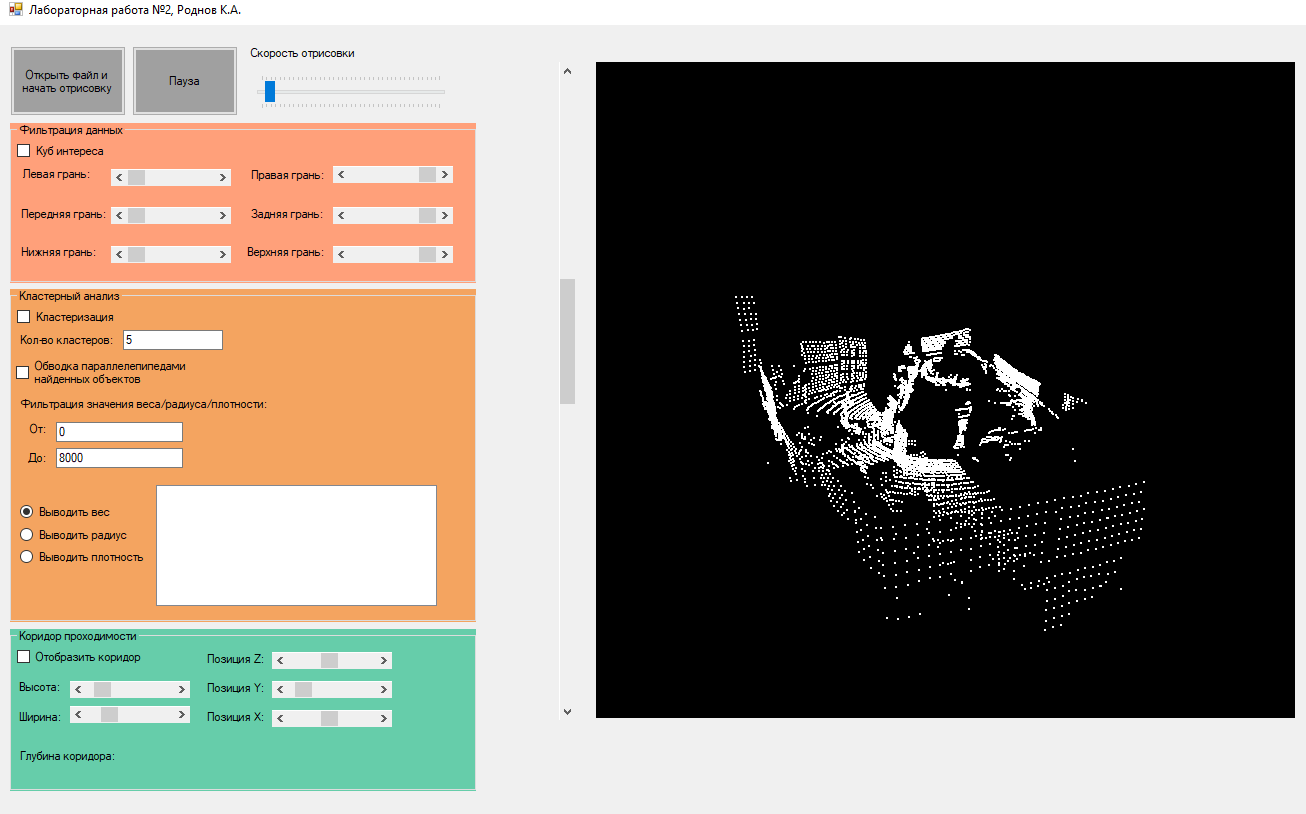


Рисунок 1 – Отображение облака точек

2) Реализована функция поворота матрицы точек, а также функция зуммирования сцены с помощью мыши и скроллбара.

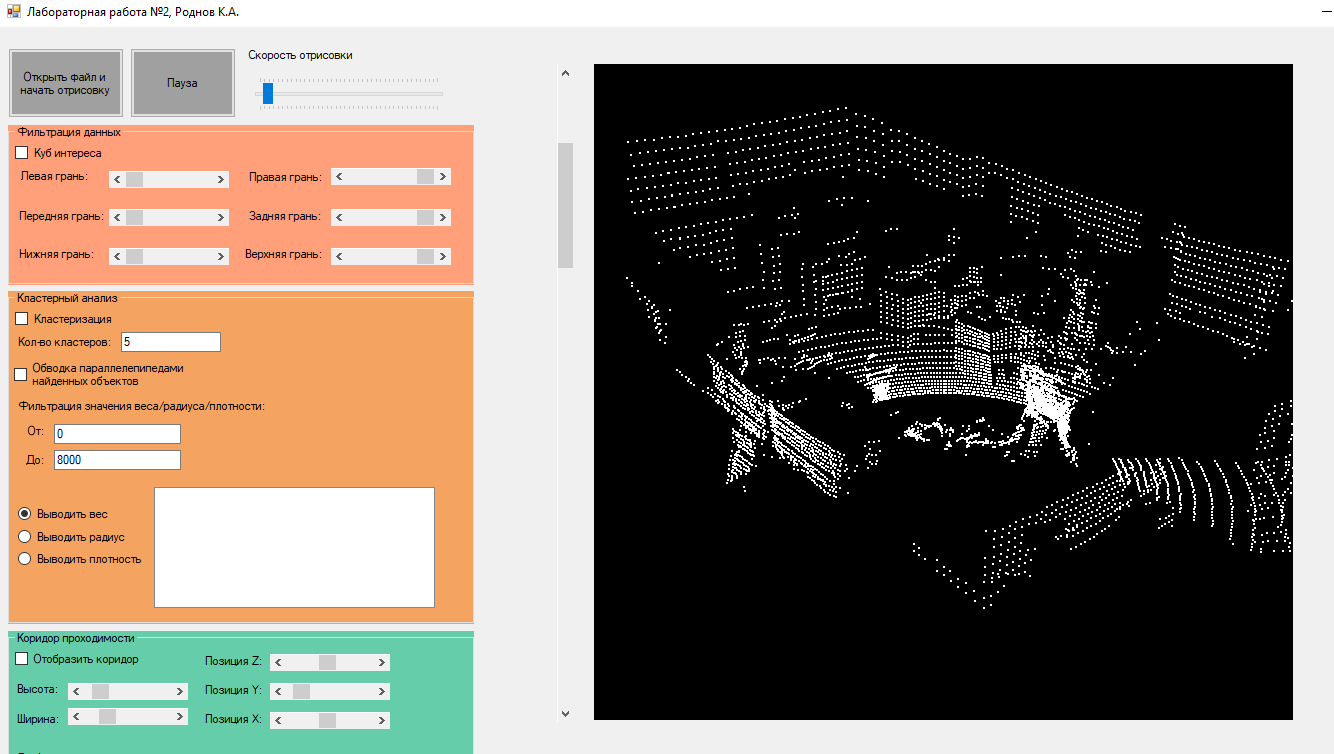


Рисунок 2 – Поворот и приближение сцены

3) Разработана функция фильтрации данных через куб интереса (грани куба отсекают ненужные области сцены, скрывая точки).

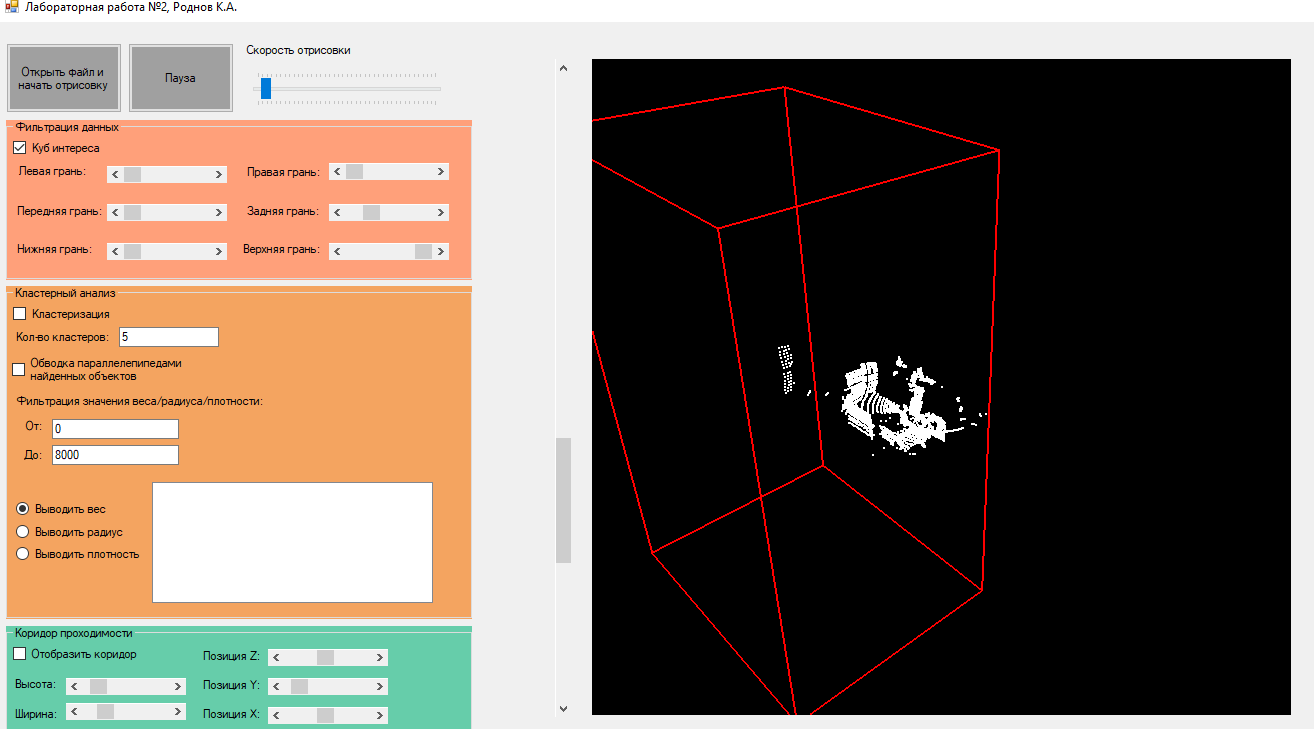


Рисунок 3 – Визуализация куба и фильтрация данных

3) Разработана функция кластерного анализа данных методом k-means, пользователь указывает количество кластеров и один из видов отображаемых параметров для дальнейшей локализации объектов (вес, радиус, плотность).

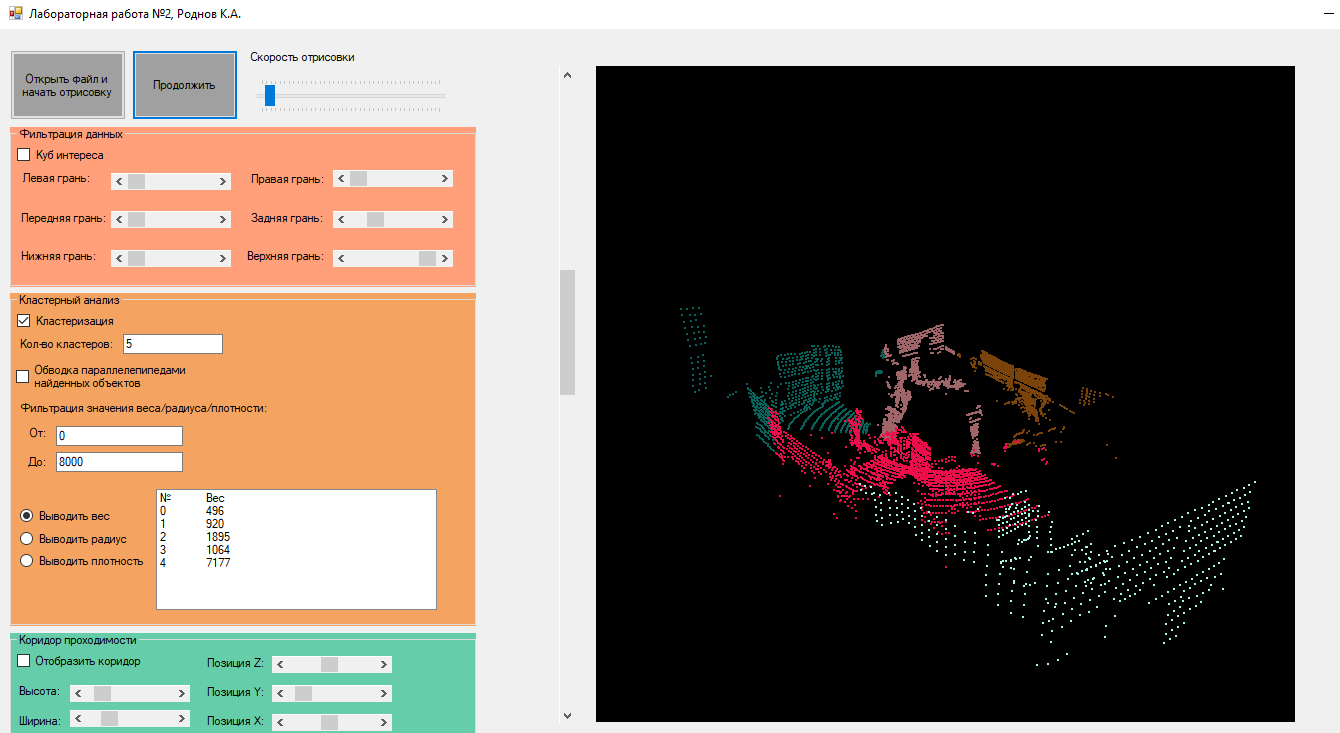


Рисунок 4 – Кластеризация методом k-means

4) Разработана функция локализации и обводки параллелепипедами найденных объектов в сцене на основе кластеризации. Пользователь может указать параметры веса, радиуса и плотности, на основе которых будет произведена локализация.

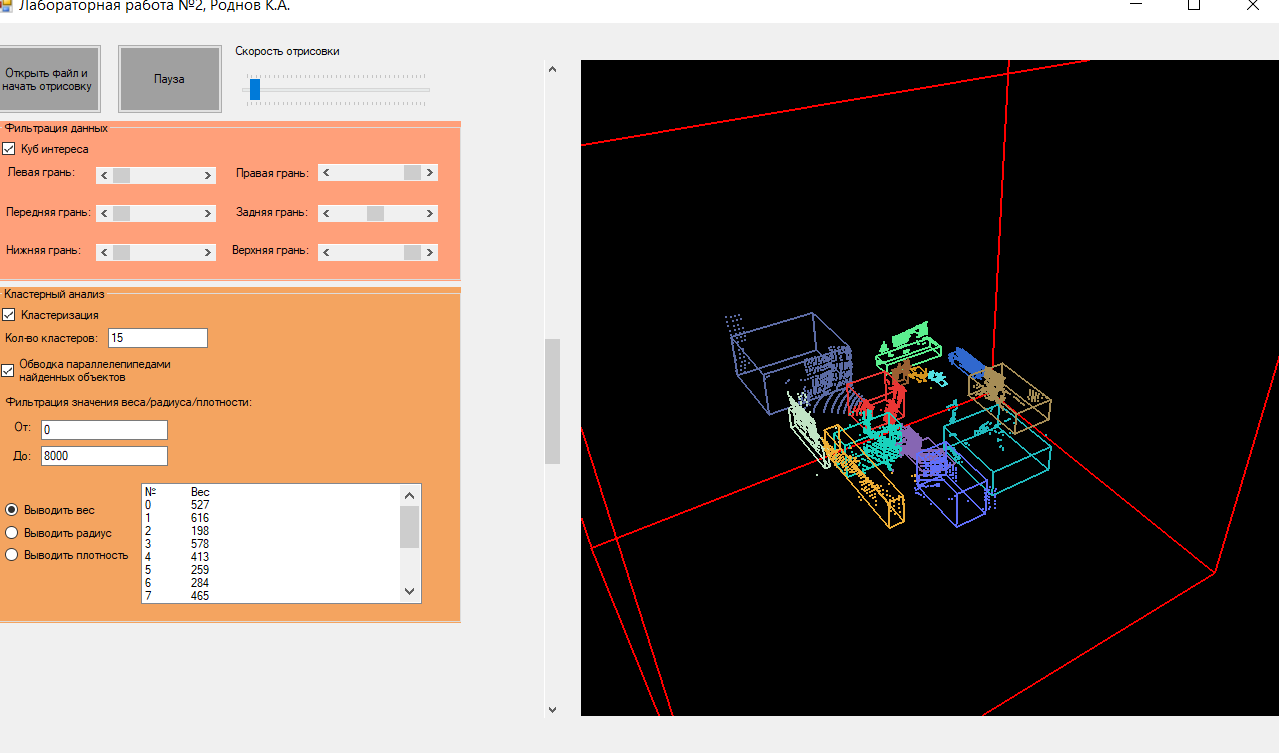


Рисунок 5 – Локализация объектов

5) Разработана функция расчёта глубины и отрисовки коридора проходимости. Пользователь задаёт размеры коридора и его координаты через скроллбары в соответствующем окне. Далее производится отсечение плоскостью куба точек в направлении взгляда на протяжённое расстояние, в извлеченном облаке производится поиск точки с минимальной координатой и в случае, если она удовлетворяет условиям, отрисовывается коридор до неё и выводится расстояние.

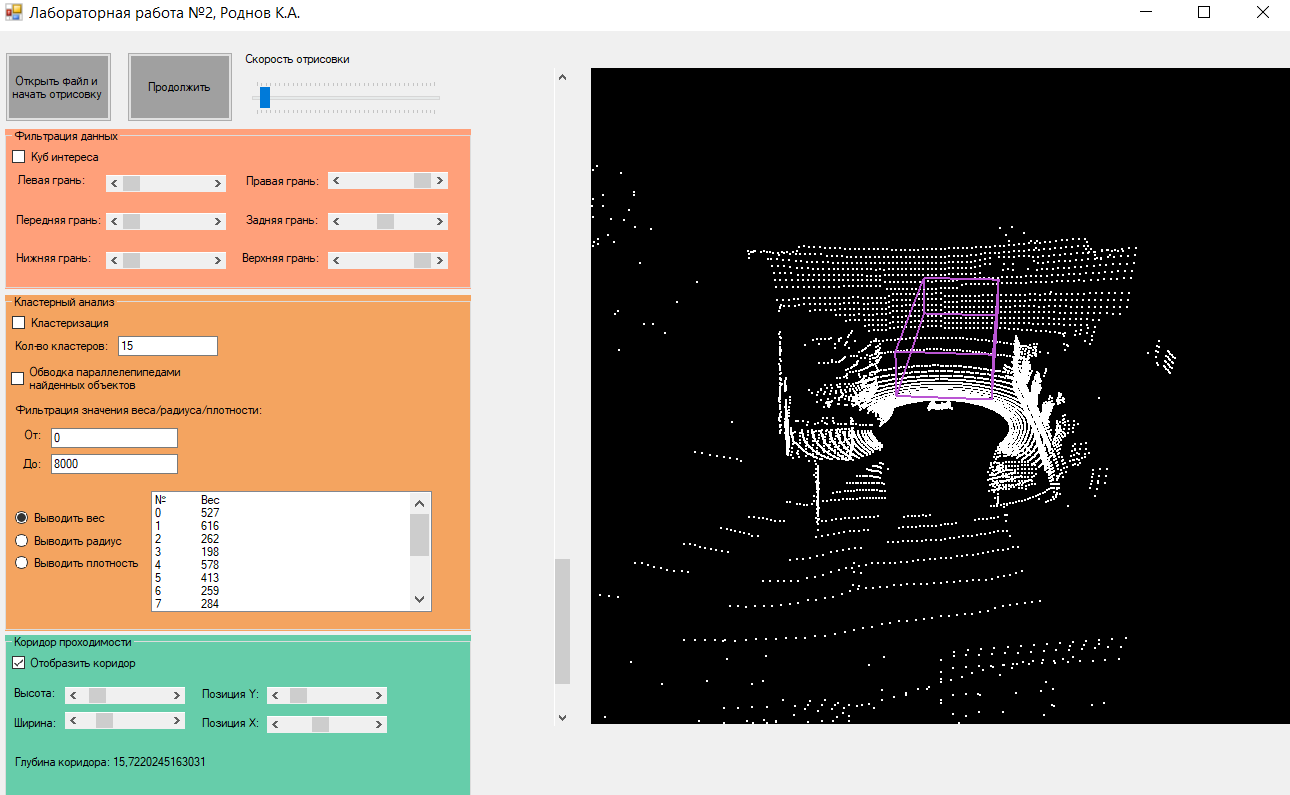


Рисунок 6 – Отображение и вычисление глубины коридора проходимости

**Вывод:** в ходе лабораторной работы было создано приложение, реализующее алгоритм считывания, визуализации и обработки потоковых данных в виде динамически обновляемого облака точек многолучевого 3D-лидара.

**Приложение А**

Листинг А1 – Основной код формы

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Windows**.**Forms**;**

**using** System**.**IO**;**

**using** System**.**Runtime**.**InteropServices**;**

**using** Microsoft**.**Win32**;**

**using** DynamicArrayLibrary**;**

**using** static System**.**Windows**.**Forms**.**VisualStyles**.**VisualStyleElement**;**

**using** System**.**Threading**;**

**using** System**.**Windows**.**Media**.**Media3D**;**

**using** SharpGL**;**

**namespace** Lab\_2

**{**

**public** **partial** class Form1 **:** Form

**{**

**public** Form1**()**

**{**

InitializeComponent**();**

SetupSinCos**();**

SetupClusterColors**();**

**}**

int rotate\_flag **=** 0**;**

volatile bool IsPaused **=** **false;**

volatile float MinAzimut **=** 0f**;**

volatile float MaxAzimut **=** 360f**;**

volatile int PlaybackSpeed **=** 1**;**

FileStream FS**;**

double**[,]** SIN\_COS\_CIRCLE **=** **new** double**[**361**,** 2**];**

double**[,]** SIN\_COS\_PITCHES **=** **new** double**[**32**,** 2**];**

double**[,,]** LIDAR\_POINTS\_CLOUD **=** **new** double**[**361**,** 32**,** 3**];**

List**<**KMeansLibrary**.**Point3D**>** CLUSTER\_DATA**;**

System**.**Drawing**.**Color**[]** CLUSTER\_COLORS**;**

double**[]** CLUSTER\_WEIGHTS**;**

double**[]** CLUSTER\_RADIUSES**;**

double**[]** CLUSTER\_DENSITIES**;**

double**[]** a**;**

int ClustersCount**;**

int count **=** 200000**;**

**public** double**[]** LASER\_PITCHES **=**

**{**

**-**30.67**,**

**-**9.33**,**

**-**29.33**,**

**-**8.00**,**

**-**28.00**,**

**-**6.66**,**

**-**26.66**,**

**-**5.33**,**

**-**25.33**,**

**-**4.00**,**

**-**24.00**,**

**-**2.67**,**

**-**22.67**,**

**-**1.33**,**

**-**21.33**,**

0.00**,**

**-**20.00**,**

1.33**,**

**-**18.67**,**

2.67**,**

**-**17.33**,**

4.00**,**

**-**16.00**,**

5.33**,**

**-**14.67**,**

6.67**,**

**-**13.33**,**

8.00**,**

**-**12.00**,**

9.33**,**

**-**10.67**,**

10.67

**};**

**private** void SetupSinCos**()**

**{**

**for** **(**int i **=** 0**;** i **<** SIN\_COS\_CIRCLE**.**GetLength**(**0**);** i**++)**

**{**

SIN\_COS\_CIRCLE**[**i**,** 0**]** **=** Math**.**Sin**(**i **\*** Math**.**PI **/** 180**);**

SIN\_COS\_CIRCLE**[**i**,** 1**]** **=** Math**.**Cos**(**i **\*** Math**.**PI **/** 180**);**

**}**

**for** **(**int i **=** 0**;** i **<** SIN\_COS\_PITCHES**.**GetLength**(**0**);** i**++)**

**{**

SIN\_COS\_PITCHES**[**i**,** 0**]** **=** Math**.**Sin**(**LASER\_PITCHES**[**i**]** **\*** Math**.**PI **/** 180**);**

SIN\_COS\_PITCHES**[**i**,** 1**]** **=** Math**.**Cos**(**LASER\_PITCHES**[**i**]** **\*** Math**.**PI **/** 180**);**

**}**

**}**

**private** void SetupClusterColors**()**

**{**

CLUSTER\_COLORS **=** **new** System**.**Drawing**.**Color**[**20**];**

Random rand **=** **new** Random**();**

**for** **(**int i **=** 0**;** i **<** 20**;** i**++)**

**{**

int R **=** rand**.**Next**(**0**,** 256**);**

int G **=** rand**.**Next**(**0**,** 256**);**

int B **=** rand**.**Next**(**0**,** 256**);**

CLUSTER\_COLORS**[**i**]** **=** System**.**Drawing**.**Color**.**FromArgb**(**R**,** G**,** B**);**

**}**

**}**

**private** void Read**()**

**{**

byte**[]** Stream **=** **new** byte**[**count**];**

List**<**byte**>** Lasers **=** **new** List**<**byte**>();**

FS**.**Read**(**Stream**,** 0**,** count**);**

**if** **(**FS**.**ReadByte**()** **>** 0**)**

**{**

**for** **(**int i **=** 0**;** i **<** Stream**.**Length **-** 1**;** i**++)**

**if** **(**Stream**[**i**]** **==** 0xFF **&&** Stream**[**i **+** 1**]** **==** 0xEE**)**

**{**

i **+=** 1**;**

**if** **(**Lasers**.**Count **!=** 0**)**

**{**

**if** **(**Lasers**.**Count **==** 98**)**

**{**

double Azimut **=** **(**Lasers**[**1**]** **\*** 256 **+** Lasers**[**0**])** **/** 100**;**

// Настройка угла (сектора)

//AnglesRangeLabel.Text = $"Диапазон углов: между {AngleFromTrackBar.Value} и {AngleToTrackBar.Value}";

**if** **(!(**Azimut **>=** MinAzimut **&&** Azimut **<=** MaxAzimut**))** **continue;**

int LaserID **=** 0**;**

Lasers**.**RemoveRange**(**0**,** 2**);**

**for** **(**int j **=** 0**;** j **<** Lasers**.**Count**;** j **+=** 3**)**

**{**

float Distance **=** **(**Lasers**[**j **+** 1**]** **\*** 256 **+** Lasers**[**j**])** **\*** 0.002f**;**

double x **=** ScaleToInterval**(**Distance **\*** **(**float**)**SIN\_COS\_CIRCLE**[(**int**)**Azimut**,** 0**],** **-**1**,** 1**);** **;**

double y **=** ScaleToInterval**(**Distance **\*** **(**float**)**SIN\_COS\_PITCHES**[**LaserID**,** 0**],** **-**1**,** 1**);**

double z **=** ScaleToInterval**(**Distance **\*** **(**float**)**SIN\_COS\_CIRCLE**[(**int**)**Azimut**,** 1**],** **-**1**,** 1**);**

LIDAR\_POINTS\_CLOUD**[(**int**)**Azimut**,** LaserID**,** 0**]** **=** x**;**

LIDAR\_POINTS\_CLOUD**[(**int**)**Azimut**,** LaserID**,** 1**]** **=** y**;**

LIDAR\_POINTS\_CLOUD**[(**int**)**Azimut**,** LaserID**,** 2**]** **=** z**;**

LaserID**++;**

**}**

**}**

Lasers **=** **new** List**<**byte**>();**

**}**

**}**

**else**

Lasers**.**Add**(**Stream**[**i**]);**

**}**

**}**

double xy\_left**;**

double xy\_right**;**

double yz\_front**;**

double yz\_back**;**

double xz\_bottom**;**

double xz\_top**;**

Color color **=** Color**.**White**;**

**private** void Drawing**()**

**{**

xy\_left **=** hScrollBar1**.**Value **/** 20d**;**

xy\_right **=** hScrollBar4**.**Value **/** 20d**;**

yz\_front **=** hScrollBar5**.**Value **/** 20d**;**

yz\_back **=** hScrollBar2**.**Value **/** 20d**;**

xz\_bottom **=** hScrollBar3**.**Value **/** 20d**;**

xz\_top **=** hScrollBar6**.**Value **/** 20d**;**

xy\_left **=** ScaleToInterval**(**xy\_left**,** **-**0.25**,** 0.5**);**

xy\_right **=** ScaleToInterval**(**xy\_right**,** **-**0.25**,** 0.5**);**

yz\_front **=** ScaleToInterval**(**yz\_front**,** **-**0.25**,** 0.5**);**

xz\_bottom **=** ScaleToInterval**(**xz\_bottom**,** **-**0.25**,** 0.5**);**

yz\_back **=** ScaleToInterval**(**yz\_back**,** **-**0.25**,** 0.5**);**

xz\_top **=** ScaleToInterval**(**xz\_top**,** **-**0.25**,** 0.5**);**

rotate\_flag **=** 1**;**

// Интерпретация кластеров как объектов

openGLControl1**.**Invalidate**();**

**}**

**private** double ScaleToInterval**(**double **value,** double min**,** double max**)**

**{**

**return** min **+** **(**max **-** min**)** **\*** **(value** **+** 1**)** **/** 2**;**

**}**

//List<int> columnCounts;

//int rowsCount;

double**[,,]** LIDAR\_POINTS\_CLOUD\_COPY **=** **new** double**[**361**,** 32**,** 3**];**

**private** void Clusterize**()**

**{**

CLUSTER\_DATA **=** **new** List**<**KMeansLibrary**.**Point3D**>();**

List**<**KMeansLibrary**.**Point3D**>** Centroids **=** **new** List**<**KMeansLibrary**.**Point3D**>();**

KMeansLibrary**.**Point3D**[]** OldCentroids**;**

ClustersCount **=** Convert**.**ToInt32**(**textBox4**.**Text**);**

**if** **(**PlanesCutCheckBox**.**Checked**)**

**{**

// Проходим по всем точкам облака

**for** **(**int row **=** 0**;** row **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**0**);** row**++)**

**{**

**for** **(**int column **=** 0**;** column **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**1**);** column**++)**

**{**

// Получаем координаты текущей точки

double x **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 0**];**

double y **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 1**];**

double z **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 2**];**

// Проверяем, находится ли точка внутри плоскости отсечения

**if** **(**z **<** yz\_back **||** z **>** yz\_front **||** y **<** xz\_bottom **||** y **>** xz\_top **||** x **<** xy\_left **||** x **>** xy\_right**)**

**{**

LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 0**]** **=** 0**;**

LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 1**]** **=** 0**;**

LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 2**]** **=** 0**;**

**}**

**else**

**{**

LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 0**]** **=** x**;**

LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 1**]** **=** y**;**

LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 2**]** **=** z**;**

**}**

**}**

**}**

**for** **(**int row **=** 0**;** row **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**0**);** row**++)**

**for** **(**int column **=** 0**;** column **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**1**);** column**++)**

**{**

double x **=** LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 0**];**

double y **=** LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 1**];**

double z **=** LIDAR\_POINTS\_CLOUD\_COPY**[**row**,** column**,** 2**];**

CLUSTER\_DATA**.**Add**(new** KMeansLibrary**.**Point3D**(**z**,** x**,** y**,** **-**1**));**

**}**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

Centroids**.**Add**(**KMeansLibrary**.**Point3D**.**RandomPoint**(**i**));**

// Разбить на кластеры

**do**

**{**

OldCentroids **=** Centroids**.**ToArray**();**

CLUSTER\_DATA **=** KMeansLibrary**.**KMeans**.**Cluster**(**Centroids**,** CLUSTER\_DATA**);**

Centroids **=** KMeansLibrary**.**KMeans**.**UpdateCentroids**(**Centroids**,** CLUSTER\_DATA**,** ClustersCount**);**

**}** **while** **(!**KMeansLibrary**.**KMeans**.**CheckForConvergence**(**OldCentroids**,** Centroids**.**ToArray**()));**

// Вычислить радиусы кластеров

CLUSTER\_RADIUSES **=** **new** double**[**ClustersCount**];**

a **=** **new** double**[**ClustersCount**];**

double**[][]** Distances **=** **new** double**[**ClustersCount**][];**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

Distances**[**i**]** **=** **new** double**[**0**];**

**for** **(**int i **=** 0**;** i **<** CLUSTER\_DATA**.**Count**;** i**++)**

**{**

double x1 **=** CLUSTER\_DATA**[**i**].**x**;**

double y1 **=** CLUSTER\_DATA**[**i**].**y**;**

double z1 **=** CLUSTER\_DATA**[**i**].**z**;**

int num **=** CLUSTER\_DATA**[**i**].**cluster**;**

double x2 **=** Centroids**[**num**].**x**;**

double y2 **=** Centroids**[**num**].**y**;**

double z2 **=** Centroids**[**num**].**z**;**

double Distance **=** Math**.**Sqrt**((**x1 **-** x2**)** **\*** **(**x1 **-** x2**)** **+** **(**y1 **-** y2**)** **\*** **(**y1 **-** y2**)** **+** **(**z1 **-** z2**)** **\*** **(**z1 **-** z2**));**

Distances**[**num**]** **=** Distances**[**num**].**Add**(**Distance**);**

**}**

**try**

**{**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

**{**

CLUSTER\_RADIUSES**[**i**]** **=** Distances**[**i**].**GetMax**();**

a**[**i**]** **=** Distances**[**i**].**GetMax**();**

**}**

// Вычислить плотности кластеров

CLUSTER\_DENSITIES **=** **new** double**[**ClustersCount**];**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

**{**

double Volume **=** 4 **/** 3d **\*** Math**.**PI **\*** CLUSTER\_RADIUSES**[**i**]** **\*** CLUSTER\_RADIUSES**[**i**];**

CLUSTER\_DENSITIES**[**i**]** **=** Distances**[**i**].**Length **/** Volume**;**

**}**

// Заполнить веса кластеров

CLUSTER\_WEIGHTS **=** **new** double**[**ClustersCount**];**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

CLUSTER\_WEIGHTS**[**i**]** **=** Distances**[**i**].**Length**;**

// Вывести данные

List**.**Items**.**Clear**();**

**if** **((**bool**)**ByWeightRadioButton**.**Checked**)**

List**.**Items**.**Add**(**"№\tВес"**);**

**if** **((**bool**)**ByRadiusRadioButton**.**Checked**)**

List**.**Items**.**Add**(**"№\tРадиус"**);**

**if** **((**bool**)**ByDensityRadioButton**.**Checked**)**

List**.**Items**.**Add**(**"№\tПлотность"**);**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

**{**

**if** **((**bool**)**ByWeightRadioButton**.**Checked**)**

List**.**Items**.**Add**(**i**.**ToString**()** **+** "\t" **+** CLUSTER\_WEIGHTS**[**i**].**ToString**());**

**if** **((**bool**)**ByRadiusRadioButton**.**Checked**)**

List**.**Items**.**Add**(**i**.**ToString**()** **+** "\t" **+** CLUSTER\_RADIUSES**[**i**].**ToString**());**

**if** **((**bool**)**ByDensityRadioButton**.**Checked**)**

List**.**Items**.**Add**(**i**.**ToString**()** **+** "\t" **+** CLUSTER\_DENSITIES**[**i**].**ToString**());**

**}**

**}**

**catch** **{** **}**

**}**

**else**

**{**

**for** **(**int row **=** 0**;** row **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**0**);** row**++)**

**for** **(**int column **=** 0**;** column **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**1**);** column**++)**

**{**

double x **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 0**];**

double y **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 1**];**

double z **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 2**];**

CLUSTER\_DATA**.**Add**(new** KMeansLibrary**.**Point3D**(**z**,** x**,** y**,** **-**1**));**

**}**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

Centroids**.**Add**(**KMeansLibrary**.**Point3D**.**RandomPoint**(**i**));**

// Разбить на кластеры

**do**

**{**

OldCentroids **=** Centroids**.**ToArray**();**

CLUSTER\_DATA **=** KMeansLibrary**.**KMeans**.**Cluster**(**Centroids**,** CLUSTER\_DATA**);**

Centroids **=** KMeansLibrary**.**KMeans**.**UpdateCentroids**(**Centroids**,** CLUSTER\_DATA**,** ClustersCount**);**

**}** **while** **(!**KMeansLibrary**.**KMeans**.**CheckForConvergence**(**OldCentroids**,** Centroids**.**ToArray**()));**

// Вычислить радиусы кластеров

CLUSTER\_RADIUSES **=** **new** double**[**ClustersCount**];**

a **=** **new** double**[**ClustersCount**];**

double**[][]** Distances **=** **new** double**[**ClustersCount**][];**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

Distances**[**i**]** **=** **new** double**[**0**];**

**for** **(**int i **=** 0**;** i **<** CLUSTER\_DATA**.**Count**;** i**++)**

**{**

double x1 **=** CLUSTER\_DATA**[**i**].**x**;**

double y1 **=** CLUSTER\_DATA**[**i**].**y**;**

double z1 **=** CLUSTER\_DATA**[**i**].**z**;**

int num **=** CLUSTER\_DATA**[**i**].**cluster**;**

double x2 **=** Centroids**[**num**].**x**;**

double y2 **=** Centroids**[**num**].**y**;**

double z2 **=** Centroids**[**num**].**z**;**

double Distance **=** Math**.**Sqrt**((**x1 **-** x2**)** **\*** **(**x1 **-** x2**)** **+** **(**y1 **-** y2**)** **\*** **(**y1 **-** y2**)** **+** **(**z1 **-** z2**)** **\*** **(**z1 **-** z2**));**

Distances**[**num**]** **=** Distances**[**num**].**Add**(**Distance**);**

**}**

**try**

**{**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

**{**

CLUSTER\_RADIUSES**[**i**]** **=** Distances**[**i**].**GetMax**();**

a**[**i**]** **=** Distances**[**i**].**GetMax**();**

**}**

// Вычислить плотности кластеров

CLUSTER\_DENSITIES **=** **new** double**[**ClustersCount**];**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

**{**

double Volume **=** 4 **/** 3d **\*** Math**.**PI **\*** CLUSTER\_RADIUSES**[**i**]** **\*** CLUSTER\_RADIUSES**[**i**];**

CLUSTER\_DENSITIES**[**i**]** **=** Distances**[**i**].**Length **/** Volume**;**

**}**

// Заполнить веса кластеров

CLUSTER\_WEIGHTS **=** **new** double**[**ClustersCount**];**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

CLUSTER\_WEIGHTS**[**i**]** **=** Distances**[**i**].**Length**;**

// Вывести данные

List**.**Items**.**Clear**();**

**if** **((**bool**)**ByWeightRadioButton**.**Checked**)**

List**.**Items**.**Add**(**"№\tВес"**);**

**if** **((**bool**)**ByRadiusRadioButton**.**Checked**)**

List**.**Items**.**Add**(**"№\tРадиус"**);**

**if** **((**bool**)**ByDensityRadioButton**.**Checked**)**

List**.**Items**.**Add**(**"№\tПлотность"**);**

**for** **(**int i **=** 0**;** i **<** ClustersCount**;** i**++)**

**{**

**if** **((**bool**)**ByWeightRadioButton**.**Checked**)**

List**.**Items**.**Add**(**i**.**ToString**()** **+** "\t" **+** CLUSTER\_WEIGHTS**[**i**].**ToString**());**

**if** **((**bool**)**ByRadiusRadioButton**.**Checked**)**

List**.**Items**.**Add**(**i**.**ToString**()** **+** "\t" **+** CLUSTER\_RADIUSES**[**i**].**ToString**());**

**if** **((**bool**)**ByDensityRadioButton**.**Checked**)**

List**.**Items**.**Add**(**i**.**ToString**()** **+** "\t" **+** CLUSTER\_DENSITIES**[**i**].**ToString**());**

**}**

**}**

**catch** **{** **}**

**}**

**}**

**private** void OpenFileData\_button\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**Timer1 **!=** **null)**

Timer1**.**Stop**();**

**if** **(**FS **!=** **null)**

FS**.**Close**();**

OpenFileDialog file **=** **new** OpenFileDialog**();**

file**.**Title **=** "Выберите файл с данными"**;**

file**.**Filter **=** "pcap файл|\*.pcap"**;**

**if** **(**file**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

FS **=** **new** FileStream**(**file**.**FileName**,** FileMode**.**Open**);**

**}**

**if** **(**FS **==** **null)**

**{**

MessageBox**.**Show**(**"Файл данных с лидара не загружен"**);**

**return;**

**}**

**else**

**{**

Timer1**.**Start**();**

**}**

**}**

**private** void PlayStopbtn\_Click\_1**(object** sender**,** EventArgs e**)**

**{**

**if** **(**IsPaused**)**

**{**

IsPaused **=** **false;**

Timer1**.**Start**();**

PlayStopbtn**.**Text **=** "Пауза"**;**

**}**

**else**

**{**

IsPaused **=** **true;**

Timer1**.**Stop**();**

PlayStopbtn**.**Text **=** "Продолжить"**;**

**}**

**}**

**private** void slPlaybeckSpeed\_Scroll**(object** sender**,** EventArgs e**)**

**{**

PlaybackSpeed **=** **(**int**)**slPlaybeckSpeed**.**Value**;**

TimeSpan t **=** **new** TimeSpan**(**0**,** 0**,** 0**,** PlaybackSpeed **/** 1000**,** PlaybackSpeed **>** 1000 **?** PlaybackSpeed **%** 1000 **:** PlaybackSpeed**);**

int x **=** int**.**Parse**(**t**.**Milliseconds**.**ToString**());**

//LidarTimer.Interval = new TimeSpan(0, 0, 0, PlaybackSpeed / 1000, PlaybackSpeed > 1000 ? PlaybackSpeed % 1000 : PlaybackSpeed);

Timer1**.**Interval **=** x**;**

**}**

**private** void LidarTimer\_Tick\_1**(object** sender**,** EventArgs e**)**

**{**

Read**();**

**if** **((**bool**)**ClustersCheckBox**.**Checked**)**

Clusterize**();**

Drawing**();**

**}**

**private** double**[][,]** GetCube**(**double xy\_left**,** double xy\_right**,** double yz\_front**,** double yz\_back**,** double xz\_bottom**,** double xz\_top**)** **=>** **new** double**[][,]**

**{**

// Левая сторона

**new** double**[,]** **{** **{**yz\_back**,** xz\_bottom**,** xy\_left**},** **{**yz\_front**,** xz\_bottom**,** xy\_left**},** **{**yz\_front**,** xz\_top**,** xy\_left**},** **{**yz\_back**,** xz\_top**,** xy\_left**}** **},**

// Задняя сторона

**new** double**[,]** **{** **{**yz\_back**,** xz\_bottom**,** xy\_right**},** **{**yz\_back**,** xz\_top**,** xy\_right**},** **{**yz\_front**,** xz\_top**,** xy\_right**},** **{**yz\_front**,** xz\_bottom**,** xy\_right**}** **},**

// Нижняя сторона

**new** double**[,]** **{** **{**yz\_back**,** xz\_bottom**,** xy\_left**},** **{**yz\_back**,** xz\_bottom**,** xy\_right**},** **{**yz\_front**,** xz\_bottom**,** xy\_right**},** **{**yz\_front**,** xz\_bottom**,** xy\_left**}** **},**

// Правая сторона

**new** double**[,]** **{** **{**yz\_back**,** xz\_bottom**,** xy\_left**},** **{**yz\_front**,** xz\_bottom**,** xy\_left**},** **{**yz\_front**,** xz\_bottom**,** xy\_right**},** **{**yz\_back**,** xz\_bottom**,** xy\_right**}** **},**

// Передняя сторона

**new** double**[,]** **{** **{**yz\_back**,** xz\_top**,** xy\_left**},** **{**yz\_front**,** xz\_top**,** xy\_left**},** **{**yz\_front**,** xz\_bottom**,** xy\_left**},** **{**yz\_back**,** xz\_bottom**,** xy\_left**}** **},**

// Верхняя сторона

**new** double**[,]** **{** **{**yz\_back**,** xz\_top**,** xy\_left**},** **{**yz\_back**,** xz\_top**,** xy\_right**},** **{**yz\_front**,** xz\_top**,** xy\_right**},** **{**yz\_front**,** xz\_top**,** xy\_left**}** **},**

**};**

List**<**Point3D**>** pointsInsideCube**;**

**private** void openGLControl1\_OpenGLDraw\_1**(object** sender**,** RenderEventArgs args**)**

**{**

OpenGL gl **=** openGLControl1**.**OpenGL**;**

gl**.**Clear**(**OpenGL**.**GL\_COLOR\_BUFFER\_BIT **|** OpenGL**.**GL\_DEPTH\_BUFFER\_BIT**);**

gl**.**LoadIdentity**();** // Сбрасываем матрицу моделирования

gl**.**MatrixMode**(**OpenGL**.**GL\_MODELVIEW**);**

gl**.**LookAt**(**0**,** 4**,** zoomValue**,** 0**,** 0**,** 0**,** 0**,** 1**,** 0**);**

gl**.**PointSize**(**2**);**

// Применяем повороты

gl**.**Rotate**(**rotationX**,** 1.0f**,** 0.0f**,** 0.0f**);**

gl**.**Rotate**(**rotationY**,** 0.0f**,** 1.0f**,** 0.0f**);**

gl**.**PushAttrib**(**OpenGL**.**GL\_ALL\_ATTRIB\_BITS**);**

gl**.**PushMatrix**();**

**if** **(**PlanesCutCheckBox**.**Checked**)**

**{**

gl**.**Translate**(**0.0f**,** 0.0f**,** 0.0f**);**

gl**.**PushMatrix**();**

gl**.**Color**(**1.0**,** 0.0**,** 0.0**);**

gl**.**LineWidth**(**2**);**

gl**.**Begin**(**OpenGL**.**GL\_LINES**);**

// Грань x = xy\_left

gl**.**Vertex**(**xy\_left**,** xz\_bottom**,** yz\_back**);**

gl**.**Vertex**(**xy\_left**,** xz\_bottom**,** yz\_front**);**

gl**.**Vertex**(**xy\_left**,** xz\_bottom**,** yz\_front**);**

gl**.**Vertex**(**xy\_left**,** xz\_top**,** yz\_front**);**

gl**.**Vertex**(**xy\_left**,** xz\_top**,** yz\_front**);**

gl**.**Vertex**(**xy\_left**,** xz\_top**,** yz\_back**);**

gl**.**Vertex**(**xy\_left**,** xz\_top**,** yz\_back**);**

gl**.**Vertex**(**xy\_left**,** xz\_bottom**,** yz\_back**);**

// Грань x = xy\_right

gl**.**Vertex**(**xy\_right**,** xz\_bottom**,** yz\_back**);**

gl**.**Vertex**(**xy\_right**,** xz\_bottom**,** yz\_front**);**

gl**.**Vertex**(**xy\_right**,** xz\_bottom**,** yz\_front**);**

gl**.**Vertex**(**xy\_right**,** xz\_top**,** yz\_front**);**

gl**.**Vertex**(**xy\_right**,** xz\_top**,** yz\_front**);**

gl**.**Vertex**(**xy\_right**,** xz\_top**,** yz\_back**);**

gl**.**Vertex**(**xy\_right**,** xz\_top**,** yz\_back**);**

gl**.**Vertex**(**xy\_right**,** xz\_bottom**,** yz\_back**);**

// Соединяем грани

gl**.**Vertex**(**xy\_left**,** xz\_bottom**,** yz\_back**);**

gl**.**Vertex**(**xy\_right**,** xz\_bottom**,** yz\_back**);**

gl**.**Vertex**(**xy\_left**,** xz\_bottom**,** yz\_front**);**

gl**.**Vertex**(**xy\_right**,** xz\_bottom**,** yz\_front**);**

gl**.**Vertex**(**xy\_left**,** xz\_top**,** yz\_front**);**

gl**.**Vertex**(**xy\_right**,** xz\_top**,** yz\_front**);**

gl**.**Vertex**(**xy\_left**,** xz\_top**,** yz\_back**);**

gl**.**Vertex**(**xy\_right**,** xz\_top**,** yz\_back**);**

gl**.**End**();**

gl**.**Flush**();**

gl**.**PopMatrix**();**

**}**

**if** **(**rotate\_flag **==** 1**)**

**{**

double**[]** XS **=** **new** double**[**0**];**

double cor\_w **=** hScrollBar8**.**Value **/** 10d**;**

double cor\_h **=** hScrollBar7**.**Value **/** 10d**;**

double cor\_z **=** hScrollBar9**.**Value **/** 10d**;**

double cor\_y **=** hScrollBar10**.**Value **/** 10d**;**

cor\_w **=** ScaleToInterval**(**cor\_w**,** **-**0.5**,** 0.5**);**

cor\_h **=** ScaleToInterval**(**cor\_h**,** **-**0.5**,** 0.5**);**

cor\_z **=** ScaleToInterval**(**cor\_z**,** **-**0.5**,** 0.5**);**

cor\_y **=** ScaleToInterval**(**cor\_y**,** **-**0.5**,** 0.5**);**

pointsInsideCube **=** **new** List**<**Point3D**>();**

// Отрисовка данных с лидара

**for** **(**int row **=** 0**;** row **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**0**);** row**++)**

**{**

**for** **(**int column **=** 0**;** column **<** LIDAR\_POINTS\_CLOUD**.**GetLength**(**1**);** column**++)**

**{**

double x **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 0**];**

double y **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 1**];**

double z **=** LIDAR\_POINTS\_CLOUD**[**row**,** column**,** 2**];**

**if** **(**PlanesCutCheckBox**.**Checked**)**

**{**

**if** **(**z **<** yz\_back **||** z **>** yz\_front **||** y **<** xz\_bottom **||** y **>** xz\_top **||** x **<** xy\_left **||** x **>** xy\_right**)**

**{**

**break;**

**}**

**}**

// Проецирование 3D точки на плоскость

**if** **((**bool**)**ClustersCheckBox**.**Checked **&&** CLUSTER\_DATA **!=** **null)**

**{**

//else

//{

int ClusterNumber **=** CLUSTER\_DATA**[**row **\*** LIDAR\_POINTS\_CLOUD**.**GetLength**(**1**)** **+** column**].**cluster**;**

color **=** CLUSTER\_COLORS**[**ClusterNumber**];**

//}

**}**

**else**

color **=** Color**.**White**;**

**if** **((**bool**)**CorridorCheckBox**.**Checked**)**

**{**

**if** **(**y **<=** cor\_y **+** cor\_h **&&** y **>=** cor\_y **&&** z **<=** cor\_z **+** cor\_w **&&** z **>=** cor\_z **&&** x **>=** **-**70 **&&** x **<=** **-**2**)**

**{**

pointsInsideCube**.**Add**(new** Point3D**(**x**,** y**,** z**));**

**}**

**}**

gl**.**Begin**(**OpenGL**.**GL\_POINTS**);**

gl**.**Color**(**color**.**R **/** 255.0**,** color**.**G **/** 255.0**,** color**.**B **/** 255.0**);**

gl**.**Vertex**(**x**,** y**,** z**);** // Отрисовываем точку

gl**.**End**();**

gl**.**Flush**();**

**}**

**}**

**if** **((**bool**)**CorridorCheckBox**.**Checked**)**

**{**

ClustersCheckBox**.**Checked **=** **false;**

InterpreteCheckBox**.**Checked **=** **false;**

Point3D pointWithMinX **=** pointsInsideCube**.**OrderBy**(**p **=>** p**.**X**).**Last**();**

**if** **(**pointWithMinX**.**X **<** **-**3 **)**

**{**

double**[][,]** CorridorVectors **=** GetCube**(**cor\_z**,** cor\_z **+** cor\_w**,** **-**2**,** pointWithMinX**.**X**,** cor\_y**,** cor\_y **+** cor\_h**);**

CorridorDepthLabel**.**Text **=** "Глубина коридора: " **+** Math**.**Abs**(**pointWithMinX**.**X**).**ToString**();**

color **=** Color**.**MediumOrchid**;**

gl**.**Translate**(**0.0f**,** 0.0f**,** 0.0f**);**

// Начинаем отрисовку обводки

gl**.**PushMatrix**();**

gl**.**LineWidth**(**2**);** // Толщина линии

gl**.**Color**(**color**.**R **/** 255.0**,** color**.**G **/** 255.0**,** color**.**B **/** 255.0**);**

gl**.**Begin**(**OpenGL**.**GL\_LINES**);**

**foreach** **(**var vertexSet **in** CorridorVectors**)**

**{**

**for** **(**int j **=** 0**;** j **<** vertexSet**.**GetLength**(**0**);** j**++)**

**{**

int nextIndex **=** **(**j **+** 1**)** **%** vertexSet**.**GetLength**(**0**);** // Индекс следующей вершины для замыкания контура

gl**.**Vertex**(**vertexSet**[**j**,** 0**],** vertexSet**[**j**,** 1**],** vertexSet**[**j**,** 2**]);**

gl**.**Vertex**(**vertexSet**[**nextIndex**,** 0**],** vertexSet**[**nextIndex**,** 1**],** vertexSet**[**nextIndex**,** 2**]);**

**}**

**}**

gl**.**End**();**

gl**.**Flush**();**

gl**.**PopMatrix**();**

**}**

//foreach (var point in pointsInsideCube)

//{

// gl.Begin(OpenGL.GL\_POINTS);

// gl.Color(color.R / 255.0, color.G / 255.0, color.B / 255.0);

// gl.Vertex(point.X, point.Y, point.Z); // Отрисовываем точку

// gl.End();

// gl.Flush();

//}

**}**

// Вычисление глубины коридора проходимости

**}**

**if** **((**bool**)**ClustersCheckBox**.**Checked **&&** **(**bool**)**InterpreteCheckBox**.**Checked**)**

**{**

groupBox3**.**Visible **=** **false;**

int countClust **=** Convert**.**ToInt32**(**textBox4**.**Text**);**

double**[][]** VectorsSums **=** **new** double**[**countClust**][];**

KMeansLibrary**.**Point3D**[][]** VectorsPoints **=** **new** KMeansLibrary**.**Point3D**[**countClust**][];**

**for** **(**int i **=** 0**;** i **<** countClust**;** i**++)**

VectorsSums**[**i**]** **=** **new** double**[**0**];**

**for** **(**int i **=** 0**;** i **<** countClust**;** i**++)**

VectorsPoints**[**i**]** **=** **new** KMeansLibrary**.**Point3D**[**0**];**

**for** **(**int i **=** 0**;** i **<** CLUSTER\_DATA**.**Count**;** i**++)**

**{**

double X **=** CLUSTER\_DATA**[**i**].**x**;**

double Y **=** CLUSTER\_DATA**[**i**].**y**;**

double Z **=** CLUSTER\_DATA**[**i**].**z**;**

int num **=** CLUSTER\_DATA**[**i**].**cluster**;**

VectorsSums**[**num**]** **=** VectorsSums**[**num**].**Add**(**X **+** Y **+** Z**);**

VectorsPoints**[**num**]** **=** VectorsPoints**[**num**].**Add**(**CLUSTER\_DATA**[**i**]);**

**}**

**for** **(**int i **=** 0**;** i **<** VectorsSums**.**GetLength**(**0**);** i**++)**

**{**

**try**

**{**

double From **=** Convert**.**ToDouble**(**ObjFromTextBox**.**Text**);**

double To **=** Convert**.**ToDouble**(**ObjToTextBox**.**Text**);**

bool ObjectValid **=** **false;**

**if** **((**bool**)**ByWeightRadioButton**.**Checked**)**

ObjectValid **=** CLUSTER\_WEIGHTS**[**i**]** **>** From **&&** CLUSTER\_WEIGHTS**[**i**]** **<** To**;**

**else** **if** **((**bool**)**ByRadiusRadioButton**.**Checked**)**

ObjectValid **=** CLUSTER\_RADIUSES**[**i**]** **>** From **&&** CLUSTER\_RADIUSES**[**i**]** **<** To**;**

**else** **if** **((**bool**)**ByDensityRadioButton**.**Checked**)**

ObjectValid **=** CLUSTER\_DENSITIES**[**i**]** **>** From **&&** CLUSTER\_DENSITIES**[**i**]** **<** To**;**

**if** **(!**ObjectValid**)** **continue;**

**}**

**catch** **{** **}**

color **=** CLUSTER\_COLORS**[**i**];**

int IndexOfMin **=** VectorsSums**[**i**].**GetIndexOfMin**();**

int IndexOfMax **=** VectorsSums**[**i**].**GetIndexOfMax**();**

KMeansLibrary**.**Point3D MinPoint **=** VectorsPoints**[**i**][**IndexOfMin**];**

KMeansLibrary**.**Point3D MaxPoint **=** VectorsPoints**[**i**][**IndexOfMax**];**

double**[][,]** ObjectVertices **=** GetCube**(**MinPoint**.**x**,** MaxPoint**.**x**,** MinPoint**.**y**,** MaxPoint**.**y**,** MinPoint**.**z**,** MaxPoint**.**z**);**

gl**.**Translate**(**0.0f**,** 0.0f**,** 0.0f**);**

// Начинаем отрисовку обводки

gl**.**PushMatrix**();**

gl**.**LineWidth**(**2**);** // Толщина линии

gl**.**Color**(**color**.**R **/** 255.0**,** color**.**G **/** 255.0**,** color**.**B **/** 255.0**);**

gl**.**Begin**(**OpenGL**.**GL\_LINES**);**

**foreach** **(**var vertexSet **in** ObjectVertices**)**

**{**

**for** **(**int j **=** 0**;** j **<** vertexSet**.**GetLength**(**0**);** j**++)**

**{**

int nextIndex **=** **(**j **+** 1**)** **%** vertexSet**.**GetLength**(**0**);** // Индекс следующей вершины для замыкания контура

gl**.**Vertex**(**vertexSet**[**j**,** 0**],** vertexSet**[**j**,** 1**],** vertexSet**[**j**,** 2**]);**

gl**.**Vertex**(**vertexSet**[**nextIndex**,** 0**],** vertexSet**[**nextIndex**,** 1**],** vertexSet**[**nextIndex**,** 2**]);**

**}**

**}**

gl**.**End**();**

gl**.**Flush**();**

gl**.**PopMatrix**();**

**}**

**}**

**else**

groupBox3**.**Visible **=** **true;**

**}**

**private** float rotationX**;**

**private** float rotationY**;**

**private** float prevMouseX**;**

**private** float prevMouseY**;**

**private** bool isRotating**;**

**private** void openGLControl1\_MouseDown**(object** sender**,** MouseEventArgs e**)**

**{**

**if** **(**e**.**Button **==** MouseButtons**.**Left**)**

**{**

isRotating **=** **true;**

prevMouseX **=** e**.**X**;**

prevMouseY **=** e**.**Y**;**

**}**

**}**

**private** void openGLControl1\_MouseMove**(object** sender**,** MouseEventArgs e**)**

**{**

**if** **(**isRotating**)**

**{**

float deltaX **=** e**.**X **-** prevMouseX**;**

float deltaY **=** e**.**Y **-** prevMouseY**;**

rotationY **+=** deltaX **\*** 0.5f**;** // Масштабируем для плавности движения

rotationX **+=** deltaY **\*** 0.5f**;**

prevMouseX **=** e**.**X**;**

prevMouseY **=** e**.**Y**;**

openGLControl1**.**Invalidate**();** // Перерисовываем OpenGL Control

**}**

**}**

**private** void openGLControl1\_MouseUp**(object** sender**,** MouseEventArgs e**)**

**{**

**if** **(**e**.**Button **==** MouseButtons**.**Left**)**

**{**

isRotating **=** **false;**

**}**

**}**

int zoomValue **=** 10**;**

**private** void vScrollBar1\_Scroll**(object** sender**,** ScrollEventArgs e**)**

**{**

zoomValue **=** vScrollBar1**.**Value**;**

// Преобразуйте это значение в масштабирование или другие параметры проекции

// Перерисуйте сцену

openGLControl1**.**Invalidate**();**

**}**

**private** void label17\_Click**(object** sender**,** EventArgs e**)**

**{**

**}**

**}**

**}**

Листинг А2 – Класс для кластеризации

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**IO**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**namespace** KMeansLibrary

**{**

class KMeans

**{**

**public** static List**<**Point3D**>** Cluster**(**List**<**Point3D**>** centroids**,** List**<**Point3D**>** data**)**

**{**

**foreach** **(**var p **in** data**)** //find closest centroid

**{**

int cent **=** **-**1**;**

float dist **=** 9999999**;**

**foreach** **(**var c **in** centroids**)**

**{**

float d **=** p**.**Distance**(**c**);**

**if** **(**d **<** dist**)**

**{**

dist **=** d**;**

cent **=** c**.**cluster**;**

**}**

**}**

p**.**cluster **=** cent**;**

**}**

**return** data**;**

**}**

**public** static bool CheckForConvergence**(**Point3D**[]** oldC**,** Point3D**[]** newC**)**

**{**

**for** **(**int i **=** 0**;** i **<** oldC**.**Length**;** i**++)**

**{**

**if** **(!**oldC**[**i**].**EqualTo**(**newC**[**i**]))**

**return** **false;**

**}**

**return** **true;**

**}**

**public** static List**<**Point3D**>** UpdateCentroids**(**List**<**Point3D**>** centroids**,** List**<**Point3D**>** data**,** int k**)**

**{**

Point3D**[]** runningTotal **=** **new** Point3D**[**k**];**

int**[]** counts **=** **new** int**[**k**];**

**for** **(**int i **=** 0**;** i **<** k**;** i**++)** //initialize, maybe replace this with LINQ later?

**{**

runningTotal**[**i**]** **=** **new** Point3D**(**0**,** 0**,** 0**,** i**);**

counts**[**i**]** **=** 0**;**

**}**

**foreach** **(**var p **in** data**)** //update centroids

**{**

runningTotal**[**p**.**cluster**].**Add**(**p**);**

counts**[**p**.**cluster**]++;**

**}**

**foreach** **(**var total **in** runningTotal**)** //find averages

**{**

total**.**Divide**(**counts**[**total**.**cluster**]);**

centroids**[**total**.**cluster**]** **=** **new** Point3D**(**total**);**

**}**

**return** centroids**;**

**}**

**public** static void PrintFile**(**List**<**Point3D**>** data**,** int round**)**

**{**

List**<**string**>** output **=** **new** List**<**string**>();**

output**.**Add**(**"x,y,z,cluster"**);**

**foreach** **(**var p **in** data**)**

**{**

output**.**Add**(**p**.**ToCSV**());**

**}**

File**.**WriteAllLines**(**"output-" **+** round **+** ".txt"**,** output**.**ToArray**());**

**}**

**public** static void PrintClusters**(**List**<**Point3D**>** centroids**,** List**<**Point3D**>** data**,** int round**)**

**{**

Console**.**WriteLine**(**"Round " **+** round**);**

**foreach** **(**var c **in** centroids**)**

**{**

Console**.**Write**(**"Cluster " **+** c**.**cluster **+** " " **+** c**.**ToString**()** **+** ": "**);**

**foreach** **(**var p **in** data**)**

**{**

**if** **(**p**.**cluster **==** c**.**cluster**)**

Console**.**Write**(**p**.**ToString**()** **+** " "**);**

**}**

Console**.**WriteLine**(**""**);**

**}**

Console**.**WriteLine**(**""**);**

**}**

**}**

**public** class Point3D

**{**

static Random r **=** **new** Random**();**

**public** double x**;**

**public** double y**;**

**public** double z**;**

**public** int cluster**;**

**public** Point3D**(**double a**,** double b**,** double c**,** int k**)**

**{**

x **=** a**;**

y **=** b**;**

z **=** c**;**

cluster **=** k**;**

**}**

**public** Point3D**(**Point3D p**)**

**{**

x **=** p**.**x**;**

y **=** p**.**y**;**

z **=** p**.**z**;**

cluster **=** p**.**cluster**;**

**}**

**public** **override** string ToString**()**

**{**

**return** "(" **+** x **+** "," **+** y **+** "," **+** z **+** ")"**;**

**}**

**public** string ToCSV**()**

**{**

**return** x **+** "," **+** y **+** "," **+** z **+** "," **+** cluster **+** "\r\n"**;**

**}**

//randomly create a 3D point with values 0-255, k is the cluster (-1 means unclustered)

**public** static Point3D RandomPoint**(**int k **=** **-**1**)**

**{**

**return** **new** Point3D**(**r**.**Next**(**0**,** 256**),** r**.**Next**(**0**,** 256**),** r**.**Next**(**0**,** 256**),** k**);**

**}**

**public** bool EqualTo**(**Point3D p**)**

**{**

**if** **(**x **==** p**.**x **&&** y **==** p**.**y **&&** z **==** p**.**z**)**

**return** **true;**

**else**

**return** **false;**

**}**

**public** float Distance**(**Point3D p**)**

**{**

float deltaX **=** **(**float**)(**x **-** p**.**x**);**

float deltaY **=** **(**float**)(**y **-** p**.**y**);**

float deltaZ **=** **(**float**)(**z **-** p**.**z**);**

**return** **(**float**)**Math**.**Sqrt**(**deltaX **\*** deltaX **+** deltaY **\*** deltaY **+** deltaZ **\*** deltaZ**);**

**}**

**public** void Add**(**Point3D p**)**

**{**

x **+=** p**.**x**;**

y **+=** p**.**y**;**

z **+=** p**.**z**;**

**}**

**public** void Divide**(**int div**)**

**{**

**if** **(**div **==** 0**)**

**{**

x **=** 0**;**

y **=** 0**;**

z **=** 0**;**

**return;**

**}**

x **/=** div**;**

y **/=** div**;**

z **/=** div**;**

**}**

**}**

**}**