**Отчет по лабораторной работе №2 по предмету «Технологии визуализации данных систем управления»**

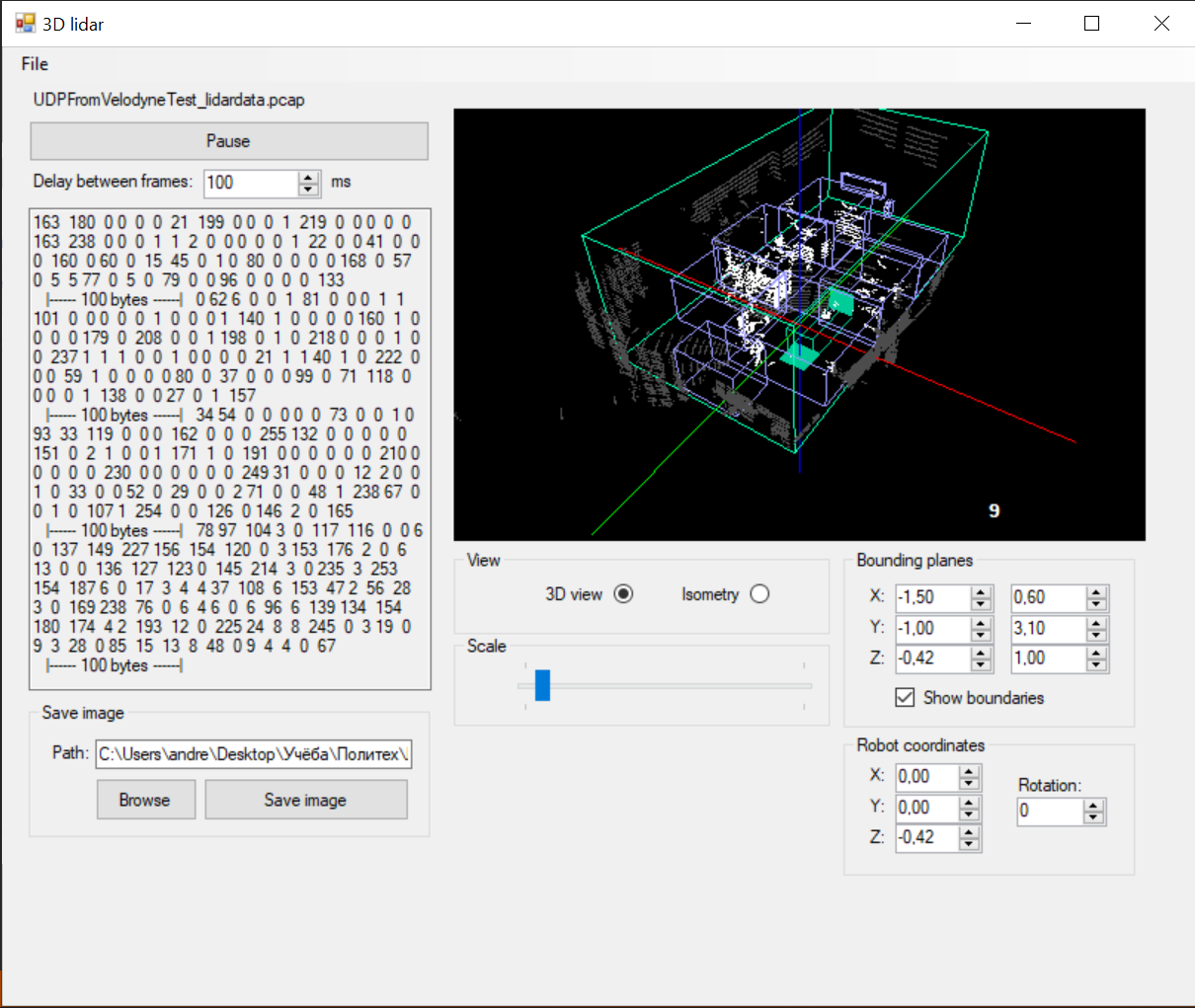
**Тема:** Обработка структурированных пространственных данных и отслеживание объектов.

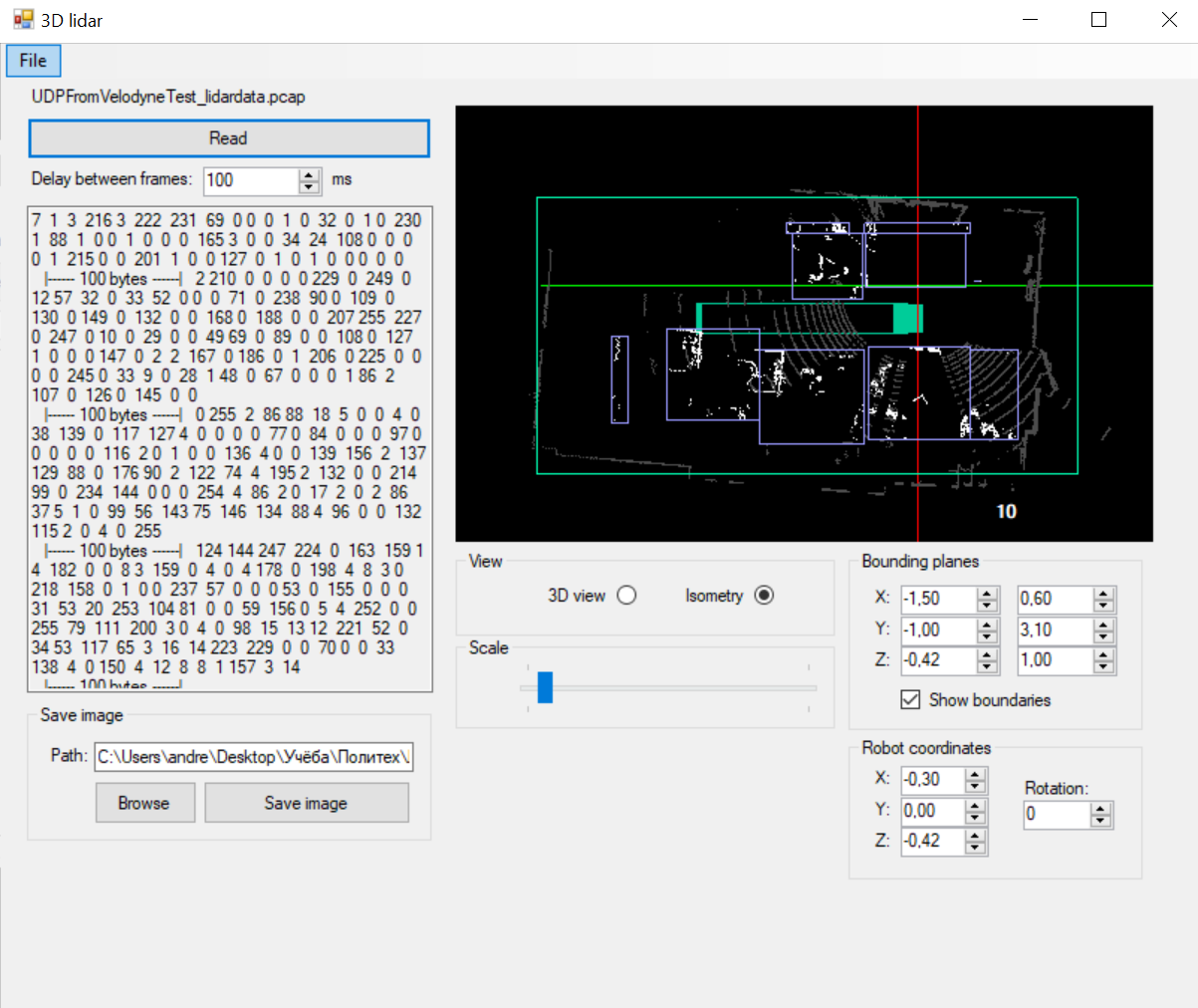
**Цель работы:** Разработать алгоритм считывания и визуализации потоковых данных в виде динамически обновляемого облака точек многолучевого 3D-лидара.

**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

**Результат работы программы:**





**Код программы:**

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.Threading;

using SharpGL;

using System.Timers;

//using SharpGL.SceneGraph.Assets;

namespace Laba\_3\_OpenGL

{

public partial class scale : Form

{

string pathVar = "C:\\Users\\andre\\Desktop\\Учёба\\Политех\\Лабораторки\\3 курс 2 семестр\\Технологии визуализации данных систем управления\\Лабораторная работа 2\\UDPFromVelodyneTest\_lidardata.pcap";

double[] tiltAngle = new double[]

{

-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,

};

bool busy = false;

bool busyText = false;

double l\_path = 4;

double[,] bp = new double[8, 3];

double xL;

double xR;

double yL;

double yR;

double zL;

double zR;

List<byte> bufText = new List<byte>();

List<byte> bufShow = new List<byte>();

double[] tiltAngleSin = new double[32];

double[] tiltAngleCos = new double[32];

RenderEventArgs args;

OpenGL gl;

List<double[]> cloudEx = new List<double[]>();

double[,] laserHistory = new double[32, 2];

int[] laserHistoryIndex = new int[32];

List<double[]> cloudIN = new List<double[]>();

List<double[]> cloudOUT = new List<double[]>();

List<double[]> cloudINShow = new List<double[]>();

List<double[]> cloudOUTShow = new List<double[]>();

int middle\_index = -1;

List<double[]> bound\_points = new List<double[]>();

List<double[]> bps = new List<double[]>();

double[] robot\_space = new double[3];

List<string> text = new List<string>();

string str = "";

byte prev = 0;

bool mouse\_down = false;

int oldValueX;

int oldValueY;

int angleX = 0;

int angleY = 0;

int angleZ = 0;

bool firstRead = true;

Thread rosa;

Thread memosa;

byte[] imageData;

public scale()

{

InitializeComponent();

}

private void Form1\_Load\_1(object sender, EventArgs e)

{

System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;

BoundariesCountT();

for (int i = 0; i < 32; i++)

{

double radAlfa = tiltAngle[i] \* Math.PI / 180.0;

tiltAngleSin[i] = Math.Sin(radAlfa);

tiltAngleCos[i] = Math.Cos(radAlfa);

laserHistory[i, 0] = -10000;

laserHistory[i, 1] = -10000;

laserHistoryIndex[i] = -1;

}

string[] aragog = pathVar.Split('\\');

labelPath.Text = aragog[aragog.Length - 1];

r\_z.Value = zlN.Value;

openGLControl1\_OpenGLDraw(sender, args);

}

private void BoundariesCountT()

{

xL = Convert.ToDouble(xlN.Value);

xR = Convert.ToDouble(xrN.Value);

yL = Convert.ToDouble(ylN.Value);

yR = Convert.ToDouble(yrN.Value);

zL = Convert.ToDouble(zlN.Value);

zR = Convert.ToDouble(zrN.Value);

bp[0, 0] = xR; bp[0, 1] = yL; bp[0, 2] = zR;

bp[1, 0] = xL; bp[1, 1] = yL; bp[1, 2] = zR;

bp[2, 0] = xL; bp[2, 1] = yL; bp[2, 2] = zL;

bp[3, 0] = xR; bp[3, 1] = yL; bp[3, 2] = zL;

bp[4, 0] = xR; bp[4, 1] = yR; bp[4, 2] = zR;

bp[5, 0] = xL; bp[5, 1] = yR; bp[5, 2] = zR;

bp[6, 0] = xL; bp[6, 1] = yR; bp[6, 2] = zL;

bp[7, 0] = xR; bp[7, 1] = yR; bp[7, 2] = zL;

}

private void Load\_file(object sender, EventArgs e)

{

System.Windows.Forms.OpenFileDialog dlg = new System.Windows.Forms.OpenFileDialog();

dlg.FileName = "Document";

dlg.DefaultExt = ".pcap";

dlg.Filter = "Text documents (.pcap)|\*.pcap";

DialogResult result = dlg.ShowDialog();

//pathBox.Text = dlg.FileName;

pathVar = dlg.FileName;

string[] aragog = pathVar.Split('\\');

labelPath.Text = aragog[aragog.Length - 1];

}

private double[] MR(double x, double y, double z, double alfa\_v)

{

alfa\_v = (alfa\_v \* Math.PI) / 180.0;

x = x \* Math.Cos(alfa\_v) + y \* Math.Sin(alfa\_v);

y = x \* -Math.Sin(alfa\_v) + y \* Math.Cos(alfa\_v);

return new double[] { x, y, z };

}

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

{

// Создаем экземпляр

gl = openGLControl1.OpenGL;

// Очистка экрана и буфера глубин

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

//gl.ClearColor(1f, 1f, 1f, 1);

// Сбрасываем модельно-видовую матрицу

gl.LoadIdentity();

double robot\_alfa = (double)alfa.Value;

double t\_x = (double)r\_x.Value;

double t\_y = (double)r\_y.Value;

double t\_z = (double)r\_z.Value;

if (notIso.Checked)

{

// Двигаем перо вглубь экрана

gl.Translate(0.0f, 0.0f, -10.0f);

gl.Rotate(angleX - 75f, angleY, angleZ + 20);

gl.Begin(OpenGL.GL\_LINES);

gl.Color(1f, 0f, 0f);

gl.Vertex(5f, 0f, 0f);

gl.Vertex(-5f, 0f, 0f);

gl.Color(0f, 1f, 0f);

gl.Vertex(0f, 5f, 0f);

gl.Vertex(0f, -5f, 0f);

gl.Color(0f, 0f, 1f);

gl.Vertex(0f, 0f, 5f);

gl.Vertex(0f, 0f, -5f);

gl.End();

double scale\_norm = Convert.ToDouble(scale\_2.Value) / 10;

if (bound\_show.Checked)

{

gl.Begin(OpenGL.GL\_LINES); //Cube

gl.Color(0.0f, 1.0f, 0.7f);

gl.Vertex(bp[0, 0] \* scale\_norm, bp[0, 1] \* scale\_norm, bp[0, 2] \* scale\_norm);

gl.Vertex(bp[1, 0] \* scale\_norm, bp[1, 1] \* scale\_norm, bp[1, 2] \* scale\_norm);

gl.Vertex(bp[1, 0] \* scale\_norm, bp[1, 1] \* scale\_norm, bp[1, 2] \* scale\_norm);

gl.Vertex(bp[2, 0] \* scale\_norm, bp[2, 1] \* scale\_norm, bp[2, 2] \* scale\_norm);

gl.Vertex(bp[2, 0] \* scale\_norm, bp[2, 1] \* scale\_norm, bp[2, 2] \* scale\_norm);

gl.Vertex(bp[3, 0] \* scale\_norm, bp[3, 1] \* scale\_norm, bp[3, 2] \* scale\_norm);

gl.Vertex(bp[3, 0] \* scale\_norm, bp[3, 1] \* scale\_norm, bp[3, 2] \* scale\_norm);

gl.Vertex(bp[0, 0] \* scale\_norm, bp[0, 1] \* scale\_norm, bp[0, 2] \* scale\_norm);

gl.Vertex(bp[2, 0] \* scale\_norm, bp[2, 1] \* scale\_norm, bp[2, 2] \* scale\_norm);

gl.Vertex(bp[6, 0] \* scale\_norm, bp[6, 1] \* scale\_norm, bp[6, 2] \* scale\_norm);

gl.Vertex(bp[1, 0] \* scale\_norm, bp[1, 1] \* scale\_norm, bp[1, 2] \* scale\_norm);

gl.Vertex(bp[5, 0] \* scale\_norm, bp[5, 1] \* scale\_norm, bp[5, 2] \* scale\_norm);

gl.Vertex(bp[3, 0] \* scale\_norm, bp[3, 1] \* scale\_norm, bp[3, 2] \* scale\_norm);

gl.Vertex(bp[7, 0] \* scale\_norm, bp[7, 1] \* scale\_norm, bp[7, 2] \* scale\_norm);

gl.Vertex(bp[0, 0] \* scale\_norm, bp[0, 1] \* scale\_norm, bp[0, 2] \* scale\_norm);

gl.Vertex(bp[4, 0] \* scale\_norm, bp[4, 1] \* scale\_norm, bp[4, 2] \* scale\_norm);

gl.Vertex(bp[4, 0] \* scale\_norm, bp[4, 1] \* scale\_norm, bp[4, 2] \* scale\_norm);

gl.Vertex(bp[5, 0] \* scale\_norm, bp[5, 1] \* scale\_norm, bp[5, 2] \* scale\_norm);

gl.Vertex(bp[5, 0] \* scale\_norm, bp[5, 1] \* scale\_norm, bp[5, 2] \* scale\_norm);

gl.Vertex(bp[6, 0] \* scale\_norm, bp[6, 1] \* scale\_norm, bp[6, 2] \* scale\_norm);

gl.Vertex(bp[6, 0] \* scale\_norm, bp[6, 1] \* scale\_norm, bp[6, 2] \* scale\_norm);

gl.Vertex(bp[7, 0] \* scale\_norm, bp[7, 1] \* scale\_norm, bp[7, 2] \* scale\_norm);

gl.Vertex(bp[7, 0] \* scale\_norm, bp[7, 1] \* scale\_norm, bp[7, 2] \* scale\_norm);

gl.Vertex(bp[4, 0] \* scale\_norm, bp[4, 1] \* scale\_norm, bp[4, 2] \* scale\_norm);

gl.End();

}

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

while (busy && playPause.Text == "Pause") ;

gl.Color(1.0f, 1.0f, 1.0f);

for (int i = 0; i < cloudINShow.Count; i++)

{

try

{

gl.Vertex(cloudINShow[i][0] \* scale\_norm, cloudINShow[i][1] \* scale\_norm, cloudINShow[i][2] \* scale\_norm);

}

catch (Exception er)

{

//MessageBox.Show(er.ToString() + "\n\r" + "i: " + i + "\n\r\n\rCloud: " + cloud.Count + "\n\rCLC: " + clc);

}

}

gl.Color(0.3f, 0.3f, 0.3f);

for (int i = 0; i < cloudOUTShow.Count; i++)

{

try

{

gl.Vertex(cloudOUTShow[i][0] \* scale\_norm, cloudOUTShow[i][1] \* scale\_norm, cloudOUTShow[i][2] \* scale\_norm);

}

catch (Exception er)

{

//MessageBox.Show(er.ToString() + "\n\r" + "i: " + i + "\n\r\n\rCloud: " + cloud.Count + "\n\rCLC: " + clc);

}

}

//if(cloudINShow.Count + cloudOUTShow.Count > 0)

// MessageBox.Show((cloudINShow.Count + cloudOUTShow.Count).ToString());

gl.End();

timeLab.Text = bps.Count.ToString();

for (int i = 0; i < bps.Count; i++)

{

double l = bps[i][3] - bps[i][0];

double w = bps[i][4] - bps[i][1];

double h = bps[i][5] - bps[i][2];

gl.Begin(OpenGL.GL\_LINES); //Cube

gl.Color(0.6f, 0.6f, 1.0f);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2]) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1] + w) \* scale\_norm, (bps[i][2]) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2]) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2] + h) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2]) \* scale\_norm);

gl.Vertex((bps[i][0] + l) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2]) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5]) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4] - w) \* scale\_norm, (bps[i][5]) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5]) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5] - h) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5]) \* scale\_norm);

gl.Vertex((bps[i][3] - l) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5]) \* scale\_norm);

gl.Vertex((bps[i][0] + l) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2] + h) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2] + h) \* scale\_norm);

gl.Vertex((bps[i][0] + l) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2] + h) \* scale\_norm);

gl.Vertex((bps[i][0] + l) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2]) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1]) \* scale\_norm, (bps[i][2] + h) \* scale\_norm);

gl.Vertex((bps[i][0]) \* scale\_norm, (bps[i][1] + w) \* scale\_norm, (bps[i][2] + h) \* scale\_norm);

gl.Vertex((bps[i][3] - l) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5] - h) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5] - h) \* scale\_norm);

gl.Vertex((bps[i][3] - l) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5] - h) \* scale\_norm);

gl.Vertex((bps[i][3] - l) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5]) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4]) \* scale\_norm, (bps[i][5] - h) \* scale\_norm);

gl.Vertex((bps[i][3]) \* scale\_norm, (bps[i][4] - w) \* scale\_norm, (bps[i][5] - h) \* scale\_norm);

gl.End();

}

double[] pb1 = NewBase(-0.15, -0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pb2 = NewBase(-0.15, 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pb3 = NewBase(0.15, 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pb4 = NewBase(0.15, -0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

gl.Begin(OpenGL.GL\_POLYGON);

gl.Color(0.0f, 0.8f, 0.6f);

gl.Vertex((pb1[0]) \* scale\_norm, (pb1[1]) \* scale\_norm, (pb1[2]) \* scale\_norm);

gl.Vertex((pb2[0]) \* scale\_norm, (pb2[1]) \* scale\_norm, (pb2[2]) \* scale\_norm);

gl.Vertex((pb3[0]) \* scale\_norm, (pb3[1]) \* scale\_norm, (pb3[2]) \* scale\_norm);

gl.Vertex((pb4[0]) \* scale\_norm, (pb4[1]) \* scale\_norm, (pb4[2]) \* scale\_norm);

gl.End();

double[] pc1 = NewBase(-0.15, l\_path - 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc2 = NewBase(0.15, l\_path - 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc3 = NewBase(0.15, l\_path - 0.15, 0.3, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc4 = NewBase(-0.15, l\_path - 0.15, 0.3, -t\_x, -t\_y, -t\_z, robot\_alfa);

gl.Begin(OpenGL.GL\_POLYGON);

gl.Vertex((pc1[0]) \* scale\_norm, (pc1[1]) \* scale\_norm, (pc1[2]) \* scale\_norm);

gl.Vertex((pc2[0]) \* scale\_norm, (pc2[1]) \* scale\_norm, (pc2[2]) \* scale\_norm);

gl.Vertex((pc3[0]) \* scale\_norm, (pc3[1]) \* scale\_norm, (pc3[2]) \* scale\_norm);

gl.Vertex((pc4[0]) \* scale\_norm, (pc4[1]) \* scale\_norm, (pc4[2]) \* scale\_norm);

gl.End();

double[] pc5 = NewBase(-0.15, 0, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc6 = NewBase(0.15, 0, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc7 = NewBase(0.15, 0, 0.3, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc8 = NewBase(-0.15, 0, 0.3, -t\_x, -t\_y, -t\_z, robot\_alfa);

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex((pc1[0]) \* scale\_norm, (pc1[1]) \* scale\_norm, (pc1[2]) \* scale\_norm);

gl.Vertex((pc5[0]) \* scale\_norm, (pc5[1]) \* scale\_norm, (pc5[2]) \* scale\_norm);

gl.Vertex((pc2[0]) \* scale\_norm, (pc2[1]) \* scale\_norm, (pc2[2]) \* scale\_norm);

gl.Vertex((pc6[0]) \* scale\_norm, (pc6[1]) \* scale\_norm, (pc6[2]) \* scale\_norm);

gl.Vertex((pc3[0]) \* scale\_norm, (pc3[1]) \* scale\_norm, (pc3[2]) \* scale\_norm);

gl.Vertex((pc7[0]) \* scale\_norm, (pc7[1]) \* scale\_norm, (pc7[2]) \* scale\_norm);

gl.Vertex((pc4[0]) \* scale\_norm, (pc4[1]) \* scale\_norm, (pc4[2]) \* scale\_norm);

gl.Vertex((pc8[0]) \* scale\_norm, (pc8[1]) \* scale\_norm, (pc8[2]) \* scale\_norm);

gl.Vertex((pc5[0]) \* scale\_norm, (pc5[1]) \* scale\_norm, (pc5[2]) \* scale\_norm);

gl.Vertex((pc6[0]) \* scale\_norm, (pc6[1]) \* scale\_norm, (pc6[2]) \* scale\_norm);

gl.Vertex((pc6[0]) \* scale\_norm, (pc6[1]) \* scale\_norm, (pc6[2]) \* scale\_norm);

gl.Vertex((pc7[0]) \* scale\_norm, (pc7[1]) \* scale\_norm, (pc7[2]) \* scale\_norm);

gl.Vertex((pc7[0]) \* scale\_norm, (pc7[1]) \* scale\_norm, (pc7[2]) \* scale\_norm);

gl.Vertex((pc8[0]) \* scale\_norm, (pc8[1]) \* scale\_norm, (pc8[2]) \* scale\_norm);

gl.Vertex((pc8[0]) \* scale\_norm, (pc8[1]) \* scale\_norm, (pc8[2]) \* scale\_norm);

gl.Vertex((pc5[0]) \* scale\_norm, (pc5[1]) \* scale\_norm, (pc5[2]) \* scale\_norm);

gl.End();

}

else

{

// Двигаем перо вглубь экрана

gl.Translate(1.5f, 0.5f, -7.0f);

gl.Rotate(0, 0, 90);

gl.Begin(OpenGL.GL\_LINES);

gl.Color(1f, 0f, 0f);

gl.Vertex(5f, 0f, 0f);

gl.Vertex(-5f, 0f, 0f);

gl.Color(0f, 1f, 0f);

gl.Vertex(0f, 5f, 0f);

gl.Vertex(0f, -5f, 0f);

gl.End();

double scale\_norm\_iso = 1.4;

if (bound\_show.Checked)

{

gl.Begin(OpenGL.GL\_LINES); //Cube

gl.Color(0.0f, 1.0f, 0.7f);

gl.Vertex(bp[0, 0] \* scale\_norm\_iso, bp[0, 1] \* scale\_norm\_iso, bp[0, 2] \* scale\_norm\_iso);

gl.Vertex(bp[1, 0] \* scale\_norm\_iso, bp[1, 1] \* scale\_norm\_iso, bp[1, 2] \* scale\_norm\_iso);

gl.Vertex(bp[1, 0] \* scale\_norm\_iso, bp[1, 1] \* scale\_norm\_iso, bp[1, 2] \* scale\_norm\_iso);

gl.Vertex(bp[5, 0] \* scale\_norm\_iso, bp[5, 1] \* scale\_norm\_iso, bp[5, 2] \* scale\_norm\_iso);

gl.Vertex(bp[5, 0] \* scale\_norm\_iso, bp[5, 1] \* scale\_norm\_iso, bp[5, 2] \* scale\_norm\_iso);

gl.Vertex(bp[4, 0] \* scale\_norm\_iso, bp[4, 1] \* scale\_norm\_iso, bp[4, 2] \* scale\_norm\_iso);

gl.Vertex(bp[4, 0] \* scale\_norm\_iso, bp[4, 1] \* scale\_norm\_iso, bp[4, 2] \* scale\_norm\_iso);

gl.Vertex(bp[0, 0] \* scale\_norm\_iso, bp[0, 1] \* scale\_norm\_iso, bp[0, 2] \* scale\_norm\_iso);

gl.End();

}

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

while (busy && playPause.Text == "Pause") ;

gl.Color(1.0f, 1.0f, 1.0f);

for (int i = 0; i < cloudINShow.Count; i++)

{

try

{

gl.Vertex(cloudINShow[i][0] \* scale\_norm\_iso, cloudINShow[i][1] \* scale\_norm\_iso, 0);

}

catch (Exception er)

{

//MessageBox.Show(er.ToString() + "\n\r" + "i: " + i + "\n\r\n\rCloud: " + cloud.Count + "\n\rCLC: " + clc);

}

}

gl.Color(0.3f, 0.3f, 0.3f);

for (int i = 0; i < cloudOUTShow.Count; i++)

{

try

{

gl.Vertex(cloudOUTShow[i][0] \* scale\_norm\_iso, cloudOUTShow[i][1] \* scale\_norm\_iso, 0);

}

catch (Exception er)

{

//MessageBox.Show(er.ToString() + "\n\r" + "i: " + i + "\n\r\n\rCloud: " + cloud.Count + "\n\rCLC: " + clc);

}

}

////if(cloudINShow.Count + cloudOUTShow.Count > 0)

//// MessageBox.Show((cloudINShow.Count + cloudOUTShow.Count).ToString());

gl.End();

//timeLab.Text = bps.Count.ToString();

for (int i = 0; i < bps.Count; i++)

{

double l = bps[i][3] - bps[i][0];

double w = bps[i][4] - bps[i][1];

double h = bps[i][5] - bps[i][2];

gl.Begin(OpenGL.GL\_LINES); //Cube

gl.Color(0.6f, 0.6f, 1.0f);

gl.Vertex((bps[i][0]) \* scale\_norm\_iso, (bps[i][1]) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][0]) \* scale\_norm\_iso, (bps[i][1] + w) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][0]) \* scale\_norm\_iso, (bps[i][1] + w) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][0] + l) \* scale\_norm\_iso, (bps[i][1] + w) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][3]) \* scale\_norm\_iso, (bps[i][4]) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][3]) \* scale\_norm\_iso, (bps[i][4] - w) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][3]) \* scale\_norm\_iso, (bps[i][4] - w) \* scale\_norm\_iso, 0);

gl.Vertex((bps[i][3] - l) \* scale\_norm\_iso, (bps[i][4] - w) \* scale\_norm\_iso, 0);

gl.End();

}

double[] pb1 = NewBase(-0.15, -0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pb2 = NewBase(-0.15, 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pb3 = NewBase(0.15, 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pb4 = NewBase(0.15, -0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

gl.Begin(OpenGL.GL\_POLYGON);

gl.Color(0.0f, 0.8f, 0.6f);

gl.Vertex((pb1[0]) \* scale\_norm\_iso, (pb1[1]) \* scale\_norm\_iso, (pb1[2]) \* scale\_norm\_iso);

gl.Vertex((pb2[0]) \* scale\_norm\_iso, (pb2[1]) \* scale\_norm\_iso, (pb2[2]) \* scale\_norm\_iso);

gl.Vertex((pb3[0]) \* scale\_norm\_iso, (pb3[1]) \* scale\_norm\_iso, (pb3[2]) \* scale\_norm\_iso);

gl.Vertex((pb4[0]) \* scale\_norm\_iso, (pb4[1]) \* scale\_norm\_iso, (pb4[2]) \* scale\_norm\_iso);

gl.End();

double[] pc1 = NewBase(-0.15, l\_path - 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc2 = NewBase(0.15, l\_path - 0.15, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc3 = NewBase(0.15, l\_path - 0.20, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc4 = NewBase(-0.15, l\_path - 0.20, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

gl.Begin(OpenGL.GL\_POLYGON);

gl.Vertex((pc1[0]) \* scale\_norm\_iso, (pc1[1]) \* scale\_norm\_iso, (pc1[2]) \* scale\_norm\_iso);

gl.Vertex((pc2[0]) \* scale\_norm\_iso, (pc2[1]) \* scale\_norm\_iso, (pc2[2]) \* scale\_norm\_iso);

gl.Vertex((pc3[0]) \* scale\_norm\_iso, (pc3[1]) \* scale\_norm\_iso, (pc3[2]) \* scale\_norm\_iso);

gl.Vertex((pc4[0]) \* scale\_norm\_iso, (pc4[1]) \* scale\_norm\_iso, (pc4[2]) \* scale\_norm\_iso);

gl.End();

double[] pc5 = NewBase(-0.15, 0, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

double[] pc6 = NewBase(0.15, 0, 0, -t\_x, -t\_y, -t\_z, robot\_alfa);

gl.Begin(OpenGL.GL\_LINES);

gl.Vertex((pc1[0]) \* scale\_norm\_iso, (pc1[1]) \* scale\_norm\_iso, (pc1[2]) \* scale\_norm\_iso);

gl.Vertex((pc5[0]) \* scale\_norm\_iso, (pc5[1]) \* scale\_norm\_iso, (pc5[2]) \* scale\_norm\_iso);

gl.Vertex((pc2[0]) \* scale\_norm\_iso, (pc2[1]) \* scale\_norm\_iso, (pc2[2]) \* scale\_norm\_iso);

gl.Vertex((pc6[0]) \* scale\_norm\_iso, (pc6[1]) \* scale\_norm\_iso, (pc6[2]) \* scale\_norm\_iso);

gl.End();

}

gl.Flush();

imageData = new byte[3 \* 700 \* 450];

gl.ReadPixels(0, 0, 700, 450, OpenGL.GL\_RGB, OpenGL.GL\_UNSIGNED\_BYTE, imageData);

}

private void TextBomb()

{

string strL = "";

byte memoryByte = 0;

try

{

string str = "";

int byteCount = 0;

int textC = 0;

while (true)

if (bufText.Count > 0)

{

//while (busyText) ;

byte bufTextClone = bufText[0];

str = str + " " + bufTextClone + " ";

if (byteCount == 100)

{

string newLine = Environment.NewLine;

str = str + newLine + " |------ " + byteCount + " bytes ------| ";

textBox.Text = str;

byteCount = -1;

}

if (str.Length > 1350)

str = str.Substring(20, str.Length - 20 - 1);

memoryByte = bufTextClone;

bufText.Clear();

byteCount++;

}

}

catch (Exception er)

{

//MessageBox.Show(er.ToString());

}

}

private double[] NewBase(double x, double y, double z, double t\_x, double t\_y, double t\_z, double alfa)

{

double alfa\_r = (alfa \* Math.PI) / 180.0;

double x\_new = x \* Math.Cos(alfa\_r) + y \* Math.Sin(alfa\_r) - t\_x;

double y\_new = x \* -Math.Sin(alfa\_r) + y \* Math.Cos(alfa\_r) - t\_y;

double z\_new = z - t\_z;

return new double[] { x\_new, y\_new, z\_new };

}

private void RosaFunc(object sender, EventArgs e)

{

try

{

FileStream fs = File.OpenRead(pathVar);

int c;

int byteCount = 0;

byte memoryByte = 0;

byte[] buf = new byte[1];

int clearCount = 0;

int rotatoinAngleByte1 = 0;

int rotatoinAngleByte2 = 0;

double rotatoinAngle = 0;

int laserNum = 0;

int byteLaserNum = 0;

double firstLaserByte = 0;

double secondLaserByte = 0;

bool volk = true;

while ((c = fs.Read(buf, 0, buf.Length)) > 0)

{

bufText.Add(buf[0]);

if (byteCount == 1)

rotatoinAngleByte1 = buf[0];

if (byteCount == 2)

{

rotatoinAngleByte2 = buf[0];

rotatoinAngle = (rotatoinAngleByte2 \* 256.0 + rotatoinAngleByte1) / 100.0;

}

if ((byteCount > 2) && (byteCount < 101))

{

if (byteLaserNum == 0)

firstLaserByte = buf[0];

if (byteLaserNum == 1)

secondLaserByte = buf[0];

if (byteLaserNum == 2)

{

double laser = (secondLaserByte \* 256.0 + firstLaserByte) \* 0.001;

cloudEx.Add(new double[] { 0, tiltAngleCos[laserNum] \* laser,

tiltAngleSin[laserNum] \* laser, laser}); // четвертой координатой записал дальность

byteLaserNum = -1;

laserNum++;

}

byteLaserNum++;

}

if ((buf[0] == 0xEE) && (memoryByte == 0xFF))

{

//bufShow = new List<byte>(bufText);

//busyText = true;

//bufShow = bufText.ToList();

//busyText = false;

//memosa = new Thread(() => TextBomb());

//memosa.Start();

//bufText.Clear();

//str = str + "--- " + byteCount + " bytes ---\n\r";

//textBox.Text = str;

//TextFunc(buf[0]);

laserNum = 0;

byteLaserNum = 0;

double max\_differential\_h = 2; // максимальный перепад по карям точки

//double max\_differential\_v = 1; // максимальный перепад по карям точки

//double max\_differential\_base = 1; // максимальный перепад базы точки

double radAngle = rotatoinAngle \* Math.PI / 180.0;

if (cloudEx.Count == 32)

{

for (int i = 1; i < 31; i++)

{

if ((Math.Abs(cloudEx[i - 1][3] - cloudEx[i][3]) > max\_differential\_h)

&& (Math.Abs(cloudEx[i + 1][3] - cloudEx[i][3]) > max\_differential\_h)

//&& (cloudEx[i - 1][3] > cloudEx[i][3])

//&& (cloudEx[i + 1][3] > cloudEx[i][3])

)

cloudEx[i][0] = -100;

if (cloudEx[i][3] < 0.5)

cloudEx[i][0] = -100;

}

for (int i = 0; i < 32; i++)

{

double newX = cloudEx[i][0] \* Math.Cos(radAngle) + cloudEx[i][1] \* Math.Sin(radAngle);

double newY = cloudEx[i][0] \* -Math.Sin(radAngle) + cloudEx[i][1] \* Math.Cos(radAngle);

double newZ = cloudEx[i][2];

if ((newX >= xL) && (newX <= xR)

&& (newY >= yL) && (newY <= yR)

&& (newZ >= zL) && (newZ <= zR))

{

if (volk == true)

{

cloudIN.Add(new double[] { newX, newY, newZ });

double t\_x = (double)r\_x.Value;

double t\_y = (double)r\_y.Value;

double t\_z = (double)r\_z.Value;

double angle = (double)alfa.Value;

double[] barrier = NewBase(newX, newY, newZ, t\_x, t\_y, t\_z, angle);

if ((barrier[0] >= -0.15) && (barrier[0] <= 0.15) &&

(barrier[2] >= 0) && (barrier[2] <= 0.3) &&

(barrier[1] >= 0))

{

l\_path = barrier[1];

//MessageBox.Show("!");

}

else

volk = false;

}

else

volk = true;

}

else

{

cloudOUT.Add(new double[] { newX, newY, newZ });

}

}

byteCount = 0;

cloudEx.Clear();

clearCount++;

try

{

if (clearCount == 2500)

{

busy = true;

//cloudShow = cloud.Select(item => (double[])item.Clone()).ToList();

cloudINShow = cloudIN.Select(item => (double[])item.Clone()).ToList();

cloudOUTShow = cloudOUT.Select(item => (double[])item.Clone()).ToList();

busy = false;

int num\_of\_points = cloudINShow.Count;

double[,] cloudInArr = new double[num\_of\_points, 3];

for (int i = 0; i < num\_of\_points; i++)

{

cloudInArr[i, 0] = cloudINShow[i][0];

cloudInArr[i, 1] = cloudINShow[i][1];

cloudInArr[i, 2] = cloudINShow[i][2];

}

//cloudInArr = Filter3D(cloudInArr, 0.03, 6);

int num\_of\_clusters = 50;

int num\_of\_iterations = 2;

int[] cluster\_array = Kmeans3D(cloudInArr, num\_of\_clusters, num\_of\_iterations);

int treshold = 1; //минимальное количество точек в кластере чтобы его помечать

for (int i = 0; i < num\_of\_clusters; i++)

{

int num\_of\_points\_in\_the\_cluster = 0;

for (int j = 0; j < num\_of\_points; j++)

if (cluster\_array[j] == i)

num\_of\_points\_in\_the\_cluster++;

//MessageBox.Show("Cluster: " + i + "\n\r" + "Num. of points: " + num\_of\_points\_in\_the\_cluster);

if (num\_of\_points\_in\_the\_cluster >= treshold)

{

double min\_x\_cluster = 10000;

double max\_x\_cluster = -10000;

double min\_y\_cluster = 10000;

double max\_y\_cluster = -10000;

double min\_z\_cluster = 10000;

double max\_z\_cluster = -10000;

for (int j = 0; j < num\_of\_points; j++)

if (cluster\_array[j] == i)

{

if (cloudINShow[j][0] < min\_x\_cluster)

min\_x\_cluster = cloudINShow[j][0];

if (cloudINShow[j][0] > max\_x\_cluster)

max\_x\_cluster = cloudINShow[j][0];

if (cloudINShow[j][1] < min\_y\_cluster)

min\_y\_cluster = cloudINShow[j][1];

if (cloudINShow[j][1] > max\_y\_cluster)

max\_y\_cluster = cloudINShow[j][1];

if (cloudINShow[j][2] < min\_z\_cluster)

min\_z\_cluster = cloudINShow[j][2];

if (cloudINShow[j][2] > max\_z\_cluster)

max\_z\_cluster = cloudINShow[j][2];

}

bound\_points.Add(new double[]

{

min\_x\_cluster, min\_y\_cluster, min\_z\_cluster,

max\_x\_cluster, max\_y\_cluster, max\_z\_cluster

});

}

}

busy = true;

bps = bound\_points.Select(item => (double[])item.Clone()).ToList();

cloudIN.Clear();

cloudOUT.Clear();

bound\_points.Clear();

busy = false;

clearCount = 0;

Thread.Sleep(Convert.ToInt32(readSpeed.Value));

}

}

catch { }

}

}

memoryByte = buf[0];

byteCount++;

}

}

catch (Exception er)

{

//MessageBox.Show(er.ToString());

}

playPause.Text = "Restart";

}

public double[,] Filter3D(double[,] x\_y\_z, double radius, int group\_size)

{

int number\_of\_points = x\_y\_z.Length / 3;

bool[] antibody = new bool[number\_of\_points];

for (int i = 0; i < number\_of\_points; i++)

{

int group = 0;

for (int j = 0; j < number\_of\_points; j++)

{

if (i != j)

{

double new\_dist = DistanceBetween3D(

x\_y\_z[i, 0], x\_y\_z[i, 1], x\_y\_z[i, 2],

x\_y\_z[j, 0], x\_y\_z[j, 1], x\_y\_z[j, 2]

);

if (new\_dist < radius)

group++;

}

}

if (group < group\_size)

antibody[i] = true;

else

antibody[i] = false;

}

int new\_num = 0;

for (int i = 0; i < number\_of\_points; i++)

if (!antibody[i])

new\_num++;

double[,] points = new double[new\_num, 3];

int new\_counter = 0;

for (int i = 0; i < number\_of\_points; i++)

if (!antibody[i])

{

points[new\_counter, 0] = x\_y\_z[i, 0];

points[new\_counter, 1] = x\_y\_z[i, 1];

points[new\_counter, 2] = x\_y\_z[i, 2];

new\_counter++;

}

return points;

}

public int[] Kmeans3D(double[,] x\_y\_z, int number\_of\_clusters, int number\_of\_iterations)

{

double[,] cluster\_centers;

//нахоим границы разброса наших точек

double maxX = -100000;

double minX = 100000;

double maxY = -100000;

double minY = 100000;

double maxZ = -100000;

double minZ = 100000;

for (int i = 0; i < x\_y\_z.Length / 3; i++)

{

if (x\_y\_z[i, 0] > maxX)

maxX = x\_y\_z[i, 0];

if (x\_y\_z[i, 0] < minX)

minX = x\_y\_z[i, 0];

if (x\_y\_z[i, 1] > maxY)

maxY = x\_y\_z[i, 1];

if (x\_y\_z[i, 1] < minY)

minY = x\_y\_z[i, 1];

if (x\_y\_z[i, 2] > maxZ)

maxZ = x\_y\_z[i, 2];

if (x\_y\_z[i, 2] < minZ)

minZ = x\_y\_z[i, 2];

}

//int number\_of\_clusters = 10;

int number\_of\_points = x\_y\_z.Length / 3;

int iter = number\_of\_iterations;

int[] cluster\_array = new int[number\_of\_points];

//разбрасывем кластеры

cluster\_centers = new double[number\_of\_clusters, 3];

Random rand = new Random();

for (int i = 0; i < number\_of\_clusters; i++)

{

cluster\_centers[i, 0] = rand.NextDouble() \* (maxX - minX) + minX;

cluster\_centers[i, 1] = rand.NextDouble() \* (maxY - minY) + minY;

cluster\_centers[i, 2] = rand.NextDouble() \* (maxZ - minZ) + minZ;

}

//рассчитываем кластеры

for (int i = 0; i < iter; i++)

{

for (int p = 0; p < number\_of\_points; p++)

{

int closest\_cluster = -1;

double min\_dist = 100000;

for (int c = 0; c < number\_of\_clusters; c++)

{

double new\_dist = DistanceBetween3D(x\_y\_z[p, 0], x\_y\_z[p, 1], x\_y\_z[p, 2],

cluster\_centers[c, 0], cluster\_centers[c, 1], cluster\_centers[c, 2]);

if (new\_dist < min\_dist)

{

closest\_cluster = c;

min\_dist = new\_dist;

}

}

cluster\_array[p] = closest\_cluster;

}

for (int c = 0; c < number\_of\_clusters; c++)

{

double middle\_x = 0;

double middle\_y = 0;

double middle\_z = 0;

double counter = 0;

for (int p = 0; p < number\_of\_points; p++)

{

if (cluster\_array[p] == c)

{

counter++;

middle\_x += x\_y\_z[p, 0];

middle\_y += x\_y\_z[p, 1];

middle\_z += x\_y\_z[p, 2];

}

}

if (counter != 0)

{

cluster\_centers[c, 0] = Convert.ToInt32(middle\_x / counter);

cluster\_centers[c, 1] = Convert.ToInt32(middle\_y / counter);

cluster\_centers[c, 2] = Convert.ToInt32(middle\_z / counter);

}

else

{

cluster\_centers[c, 0] = -10;

cluster\_centers[c, 1] = -10;

cluster\_centers[c, 2] = -10;

}

}

}

return cluster\_array;

}

public void ShowArray1D(int[] array, int period, string name)

{

string str = name + "\n\r";

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

{

str += array[i] + " ";

if ((i % period == 0) && (i != 0))

str += "\n\r";

}

MessageBox.Show(str);

}

public double DistanceBetween3D(double x1, double y1, double z1, double x2, double y2, double z2)

{

return Math.Sqrt(Math.Pow(x2 - x1, 2) + Math.Pow(y2 - y1, 2) + Math.Pow(z2 - z1, 2));

}

private void OpenGLDraw\_func(object sender, RenderEventArgs args)

{

openGLControl1\_OpenGLDraw(sender, args);

}

private void Scale\_change(object sender, EventArgs e)

{

openGLControl1\_OpenGLDraw(sender, args);

}

private void KeyDown(object sender, KeyEventArgs e)

{

}

private void MouseDown(object sender, MouseEventArgs e)

{

mouse\_down = true;

oldValueX = e.X;

oldValueY = e.Y;

}

private void MouseUp(object sender, MouseEventArgs e)

{

mouse\_down = false;

}

private void MouseMove(object sender, MouseEventArgs e)

{

if (mouse\_down)

{

angleZ += Convert.ToInt32((e.X - oldValueX) / 2.0);

angleX += Convert.ToInt32((e.Y - oldValueY) / 2.0);

openGLControl1\_OpenGLDraw(sender, args);

oldValueX = e.X;

oldValueY = e.Y;

}

}

private void ReadDump(object sender, EventArgs e)

{

if (playPause.Text == "Read")

{

if (firstRead)

{

rosa = new Thread(() => RosaFunc(sender, e));

memosa = new Thread(() => TextBomb());

rosa.Start();

memosa.Start();

firstRead = false;

}

else

{

rosa.Resume();

memosa.Resume();

}

playPause.Text = "Pause";

}

else

if (playPause.Text == "Pause")

{

rosa.Suspend();

memosa.Suspend();

playPause.Text = "Read";

}

else

if (playPause.Text == "Restart")

{

rosa.Abort();

memosa.Abort();

firstRead = true;

playPause.Text = "Read";

}

}

private void BrowseSave(object sender, EventArgs e)

{

System.Windows.Forms.FolderBrowserDialog dlg = new System.Windows.Forms.FolderBrowserDialog();

DialogResult result = dlg.ShowDialog();

//pathBox.Text = dlg.FileName;

pathSave.Text = dlg.SelectedPath;

}

private void SaveSave(object sender, EventArgs e)

{

Bitmap bit = new Bitmap(450, 700);

int arrCounter = 0;

for (int i = 0; i < 450; ++i)

{

for (int j = 0; j < 700; ++j)

{

bit.SetPixel(i, j, Color.FromArgb(imageData[arrCounter], imageData[arrCounter + 1], imageData[arrCounter + 2]));

arrCounter += 3;

}

}

Bitmap bit2 = new Bitmap(560, 360);

for (int i = 0; i < 270; ++i)

{

for (int j = 0; j < 420; ++j)

{

Color c = bit.GetPixel(i,j);

int r = c.R;

int g = c.G;

int b = c.B;

bit2.SetPixel(j, 270 - i, Color.FromArgb(r, g, b));

}

}

bit2.Save(pathSave.Text + "//image.png");

}

private void Form1\_ClosingEvent(object sender, FormClosingEventArgs e)

{

try

{

rosa.Resume();

memosa.Resume();

}

catch { }

try

{

rosa.Abort();

memosa.Abort();

}

catch { }

}

private void BoundariesCount(object sender, EventArgs e)

{

BoundariesCountT();

openGLControl1\_OpenGLDraw(sender, args);

}

private void ChangeView(object sender, EventArgs e)

{

openGLControl1\_OpenGLDraw(sender, args);

}

private void Rob\_rot(object sender, EventArgs e)

{

//double robot\_alfa = ((double)alfa.Value \* Math.PI) / 180.0;

//double t\_x = ((double)r\_x.Value + 0.15);

//double t\_y = ((double)r\_y.Value + 0.15);

//double t\_z = ((double)r\_z.Value);

////t\_x = t\_x \* Math.Cos(robot\_alfa) + t\_y \* Math.Sin(robot\_alfa);

////t\_y = -t\_x \* Math.Sin(robot\_alfa) + t\_y \* Math.Cos(robot\_alfa);

//MessageBox.Show(t\_x + "\n\r" + t\_y + "\n\r" + t\_z);

}

}

}