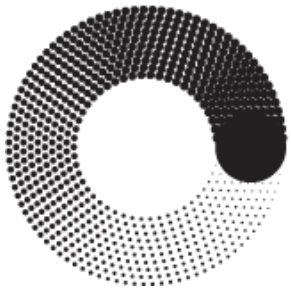
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



МОСКОВСКИЙ ПОЛИТЕХ

Кафедра СМАРТ технологий

Лабораторная работа №3
«Аппаратная поддержка отображения пространственных данных»

По дисциплине «Технологии визуализации данных систем управления»

Группа	201-325
Студент	Холодилов И.В.
Дата	24.05.2023
Преподаватель	Идиатулов Т.Т.

2023

Цель работы

Подготовить приложение на языке C# для сбора, статистической обработки и визуализации данных с использованием библиотеки OpenGL (через обертку SharpGL)

Задачи

- Реализовать загрузку набора данных, заданных как тройки чисел (X, Y, Z) из файлов формата CSV (разделитель – точка с запятой) и генерацию заданного (через текстовое поле) количества случайных точек, где X,Y и Z – равномерно распределенные случайные величины на диапазоне [-1 ÷ 1].

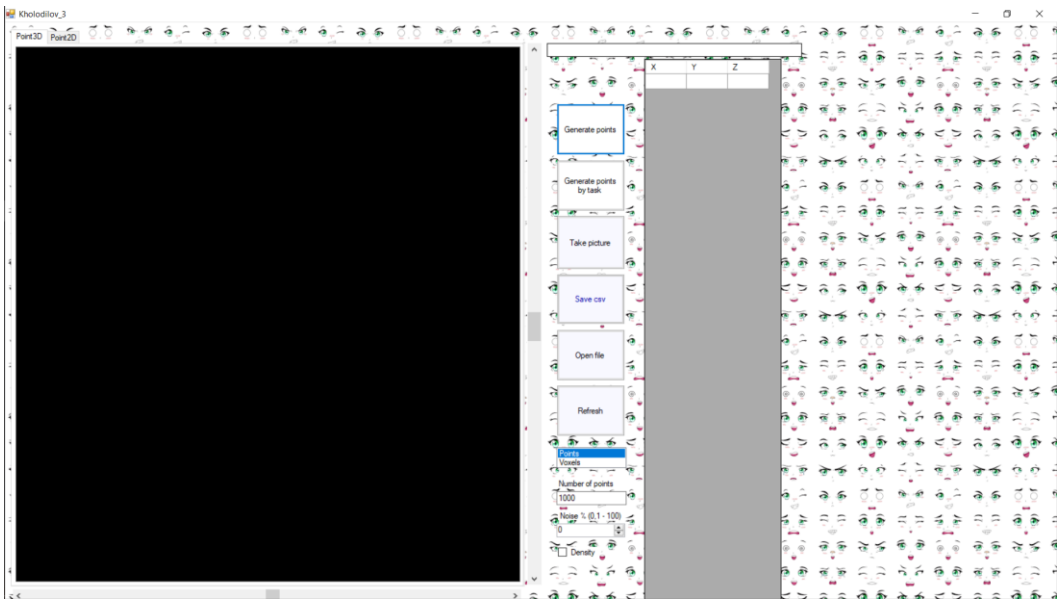


Рисунок 1 – Form

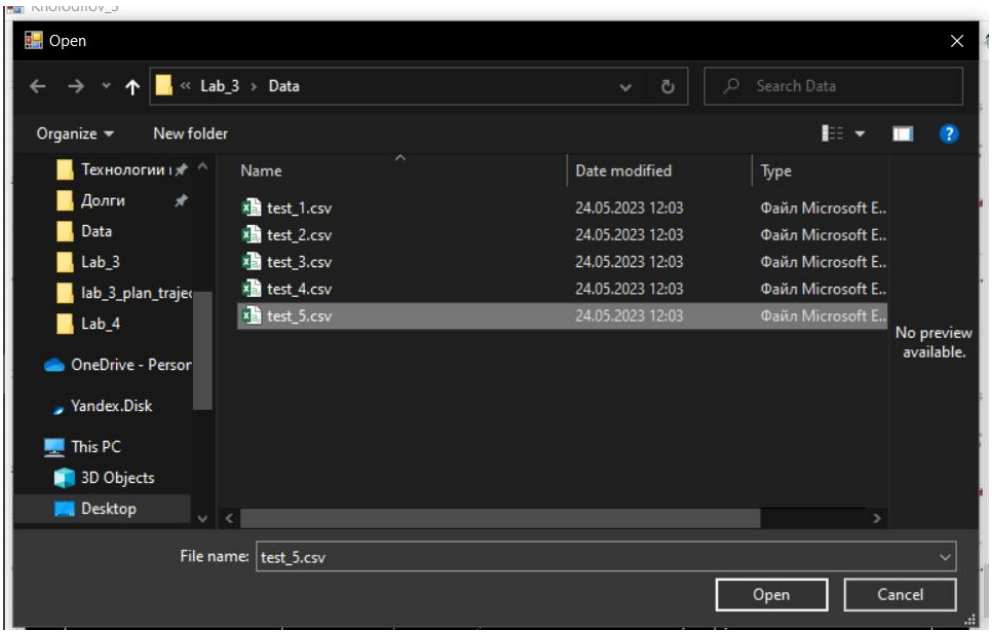


Рисунок 2 - Открытие данных

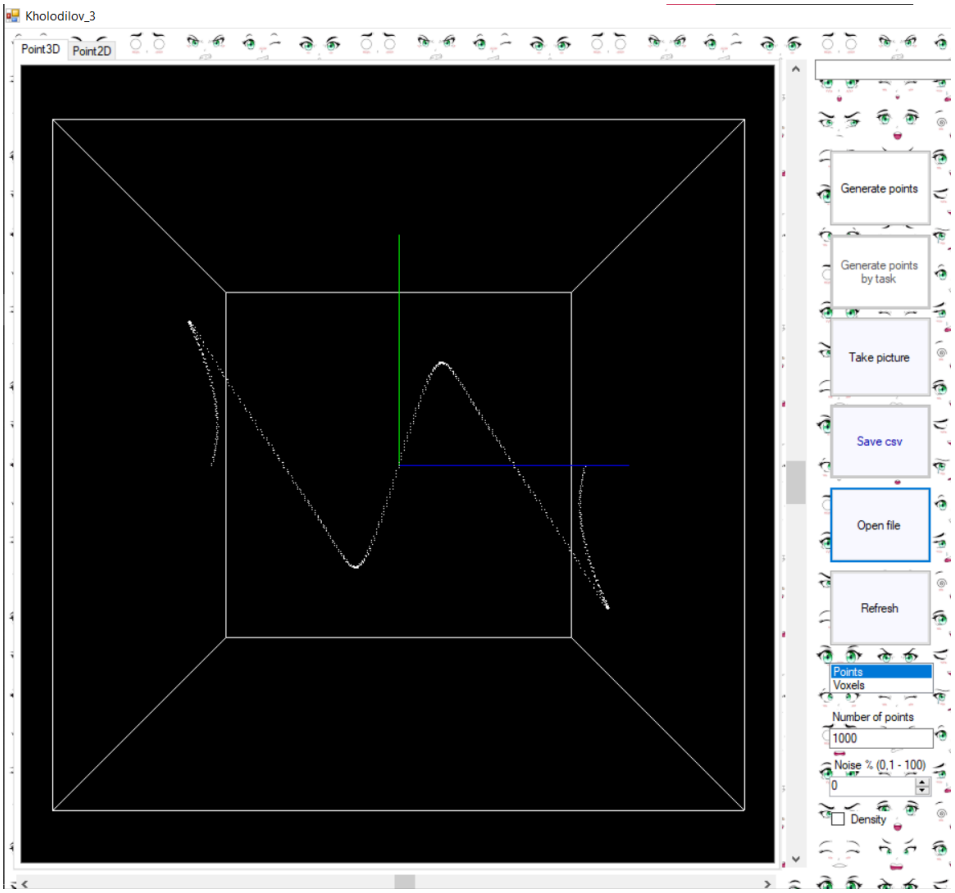


Рисунок 3 - Отображение данных из файла

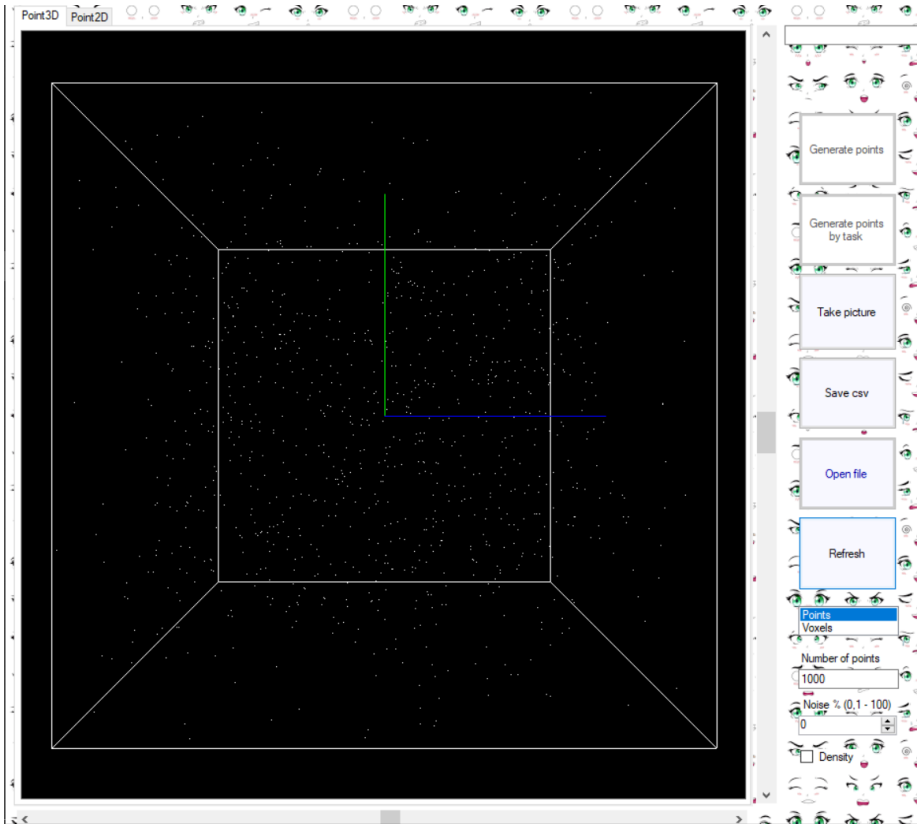


Рисунок 4 - Генерация рандомных значений и их отображение

- Выполнить статистический анализ набора точек, выполнив построение частотной диаграммы (гистограммы) 10x10 ячеек в координатной плоскости XY.

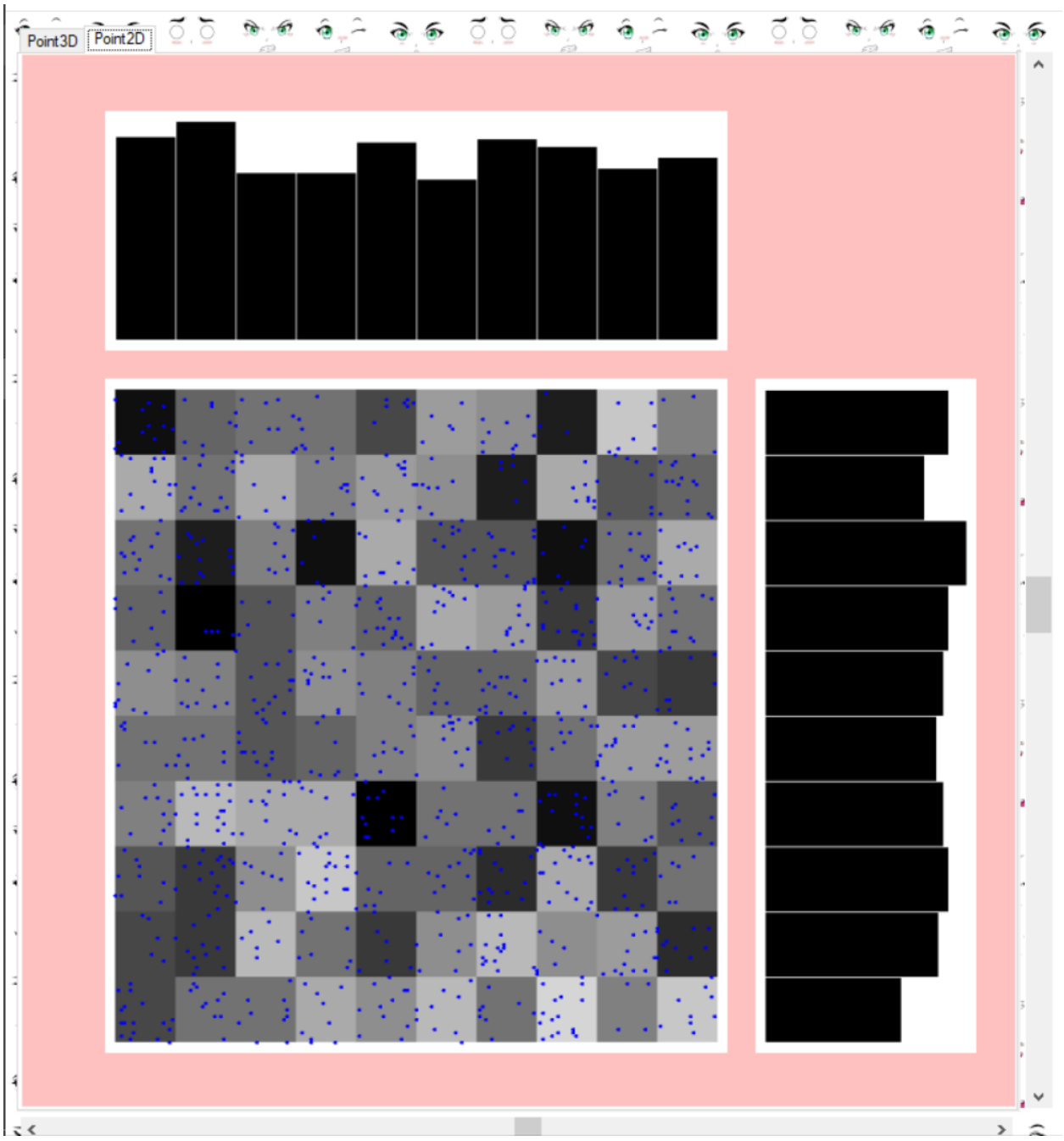


Рисунок 5 - Отображение гистограмм

- Выполнить статистический анализ набора точек, выполнив расчет плотности заполнения вокселей (voxel) как пространственной матрицы 10x10x10 вокселей (пространственных ячеек). Разработать систему отображения данных в виде пространства вокселей, значение плотности заполнения каждого из которых отображать размером кубика, помещенного в центр вокселя.

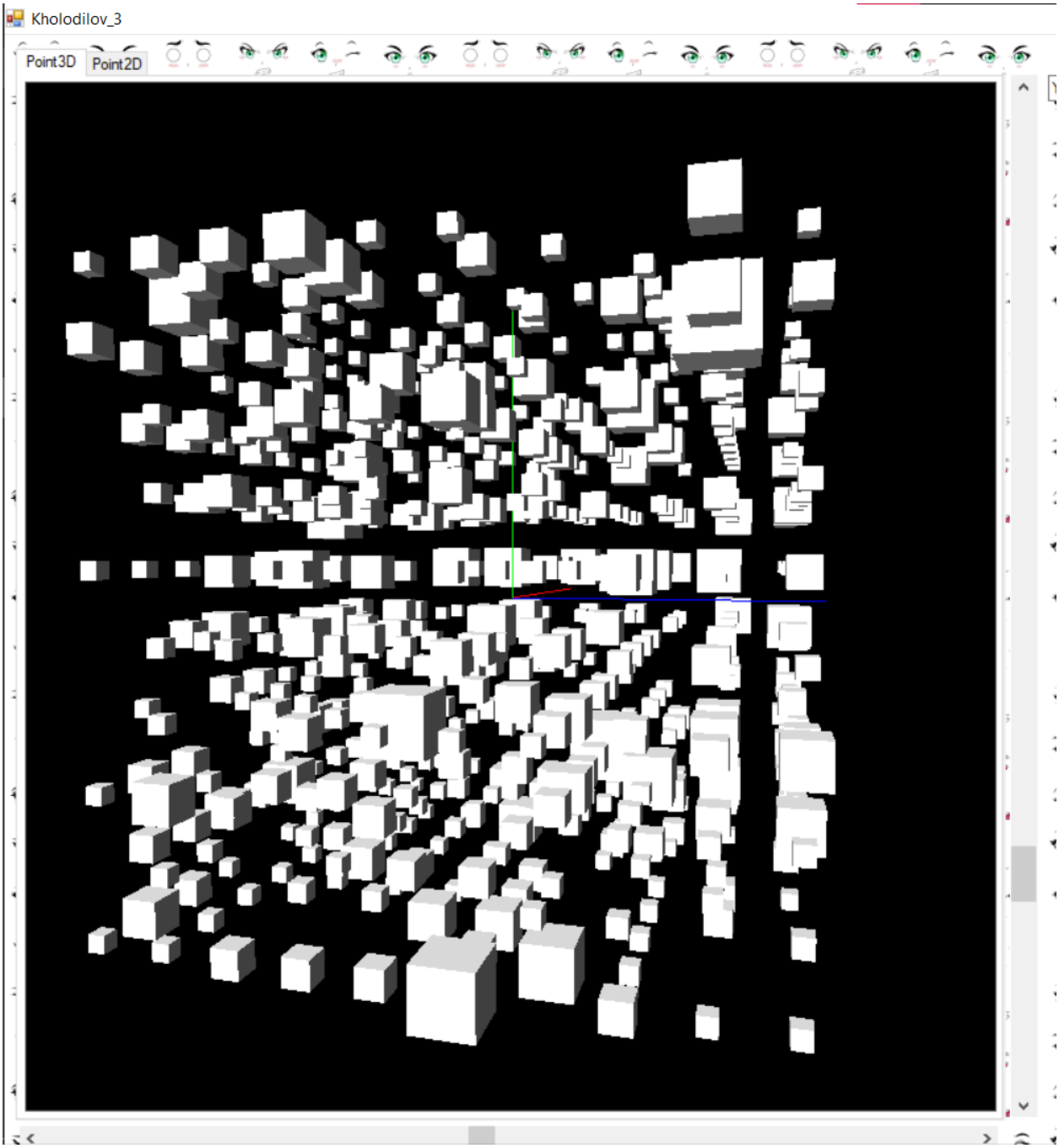


Рисунок 6 - Отрисовка вокселей

- Разработать систему отображения данных в виде облака точек средствами OpenGL (SharpGL) с указанием степени прореживания (отображать с шагом по номеру) и функцией анимации со сдвигом по номеру отображения. Разработать систему отображения гистограммы распределения точек в виде поверхности, где координаты вычисляются следующим образом: координаты X и Y соответствуют координатам ячеек (центрам) гистограммы, а координата Z вычисляется как доля частоты попадания точек в данную ячейку по отношению к самому большому значению среди всех ячеек.

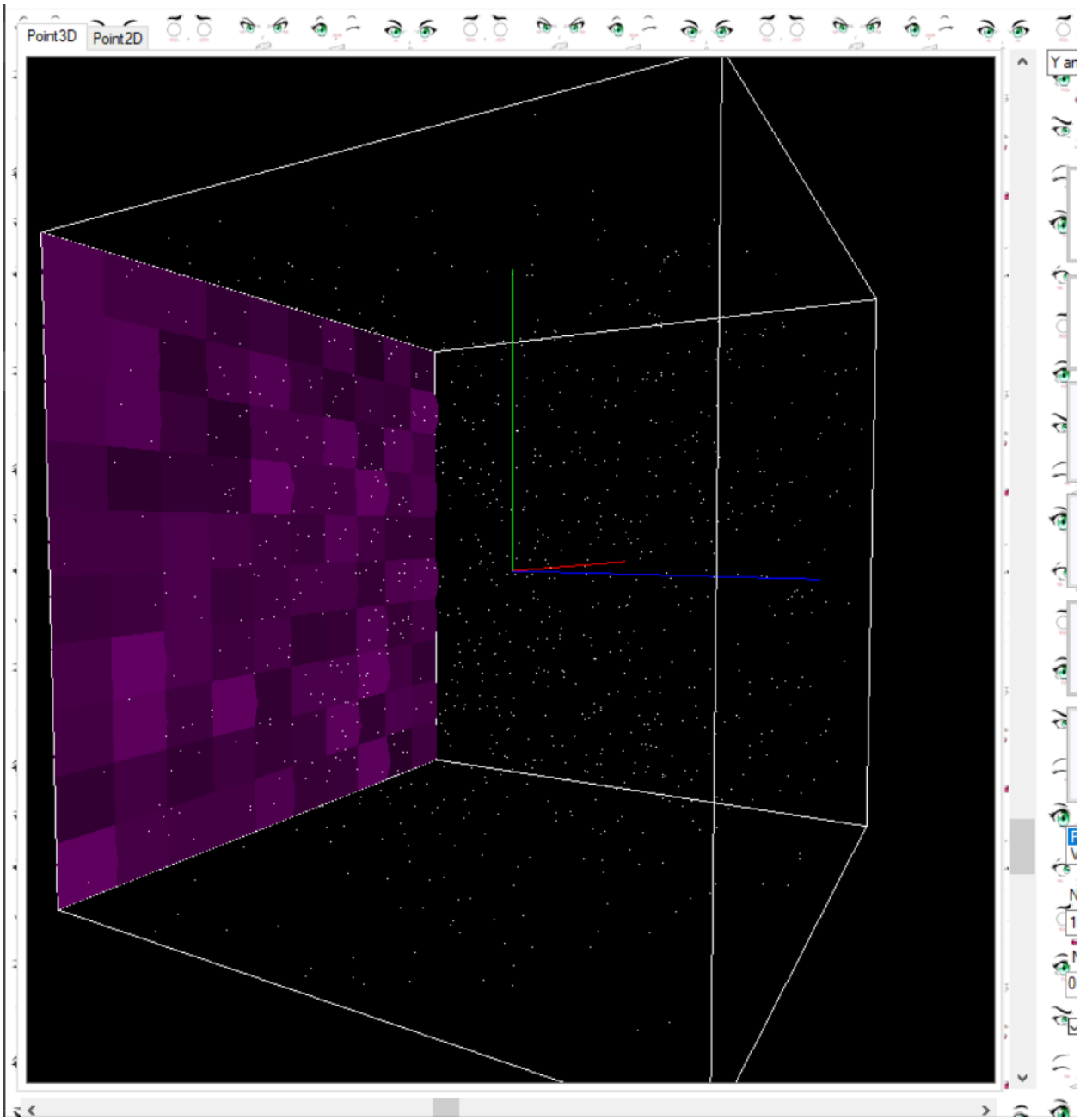


Рисунок 7 - Отрисовка гистограммы на плоскости

- Реализовать систему поворота базовой системы координат при отображении данных с помощью матрицы трансляции-поворота, управляемого позиционным манипулятором (мышью). Реализовать отображение системы координат и ребер описывающего куба (стороны: -1 и 1 по каждой координате)

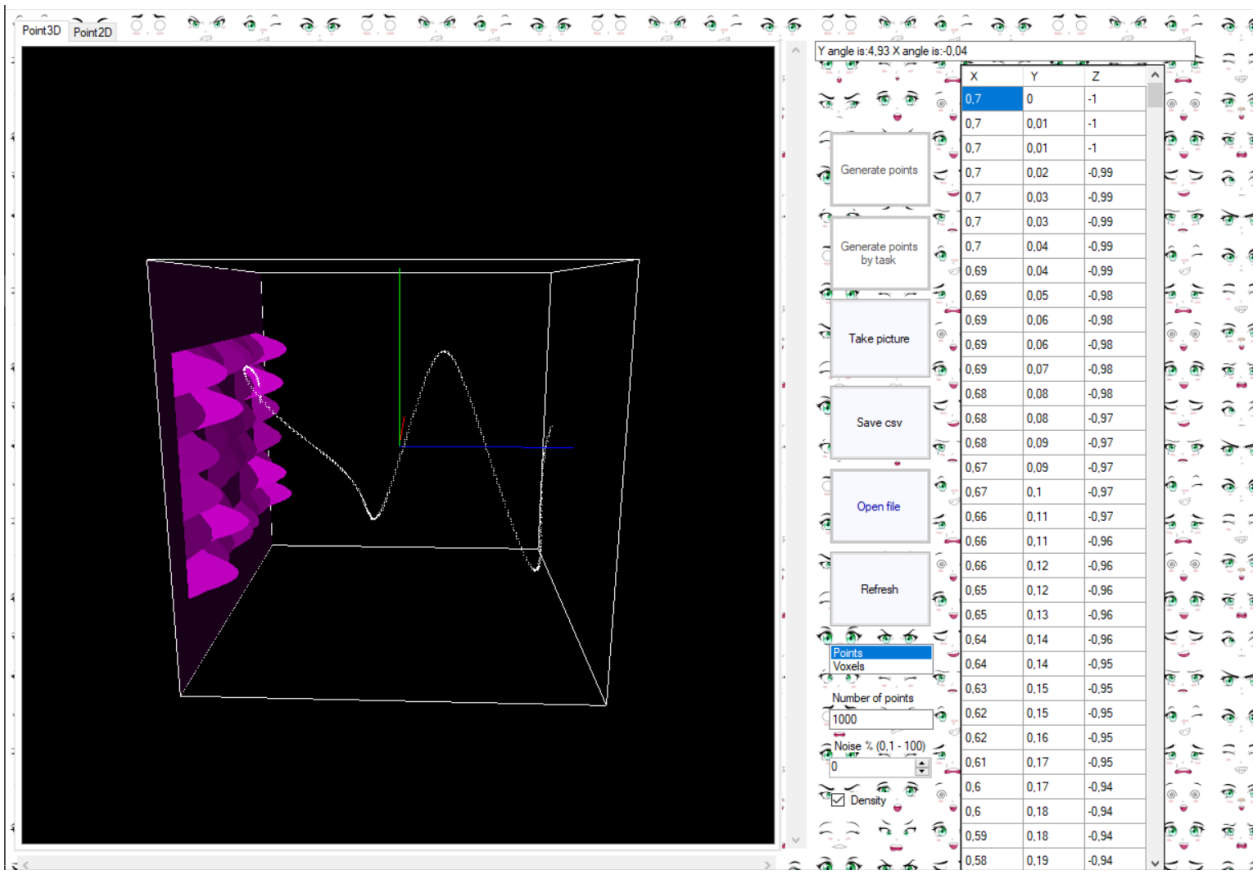


Рисунок 8 - Отображение поворота камеры

- Реализовать сохранение полученного изображения в файл.

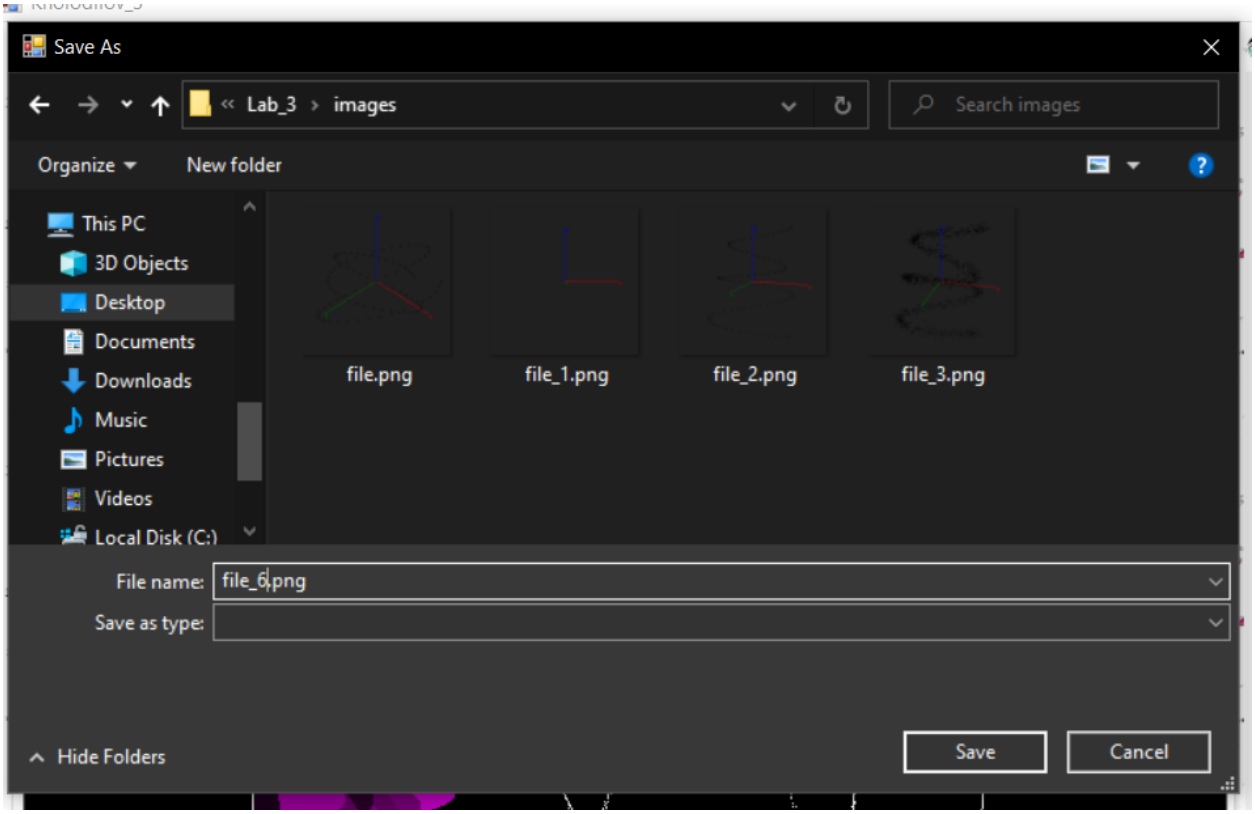


Рисунок 9 - Сохранение изображения

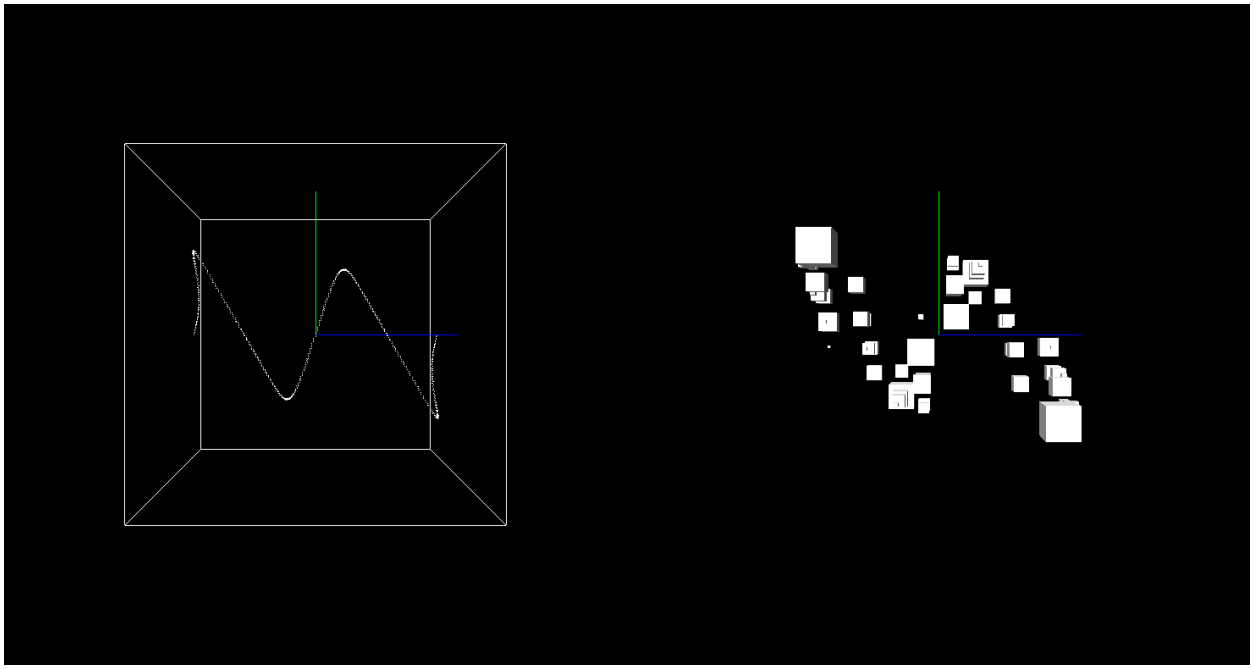


Рисунок 10 - Сохраненное изображение

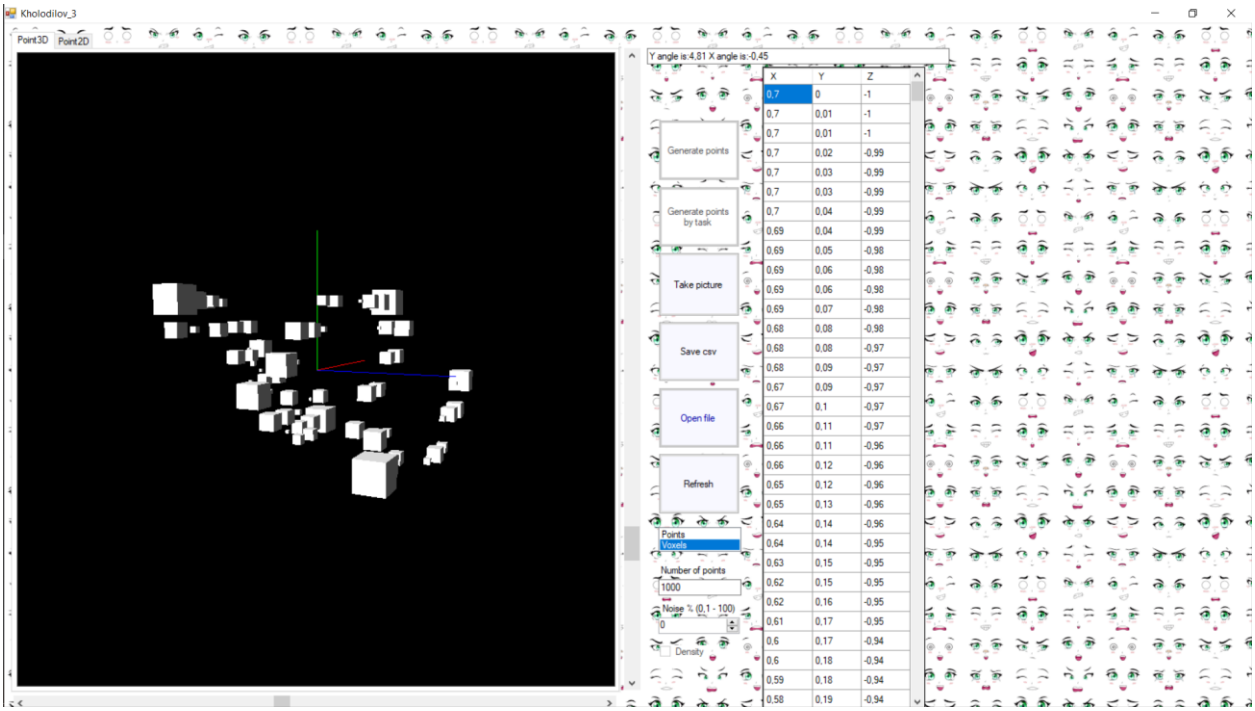


Рисунок 11 - Финальный вид программы

Вывод

В ходе создания программы было написанно ПО для отображения трехмерных данных с использованием библиотеки OpenGL.

Приложение А

Листинг А-1 – программный код:

Приложение А

Листинг А-1 – программный код:

```
using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using OpenCvSharp;
using SharpGL;
using System.Drawing.Drawing2D;
using System.IO;
using System.Drawing.Text;
using System.Runtime.InteropServices;

namespace Lab_3
{
   public partial class Form1 : Form
   {
      OpenGL gl;
      Random random = new Random();

      bool is_noise_change = false;
      bool is_generate_task = false;
      bool is_rotate_change = false;
      bool is_voxel_clear = true;
      bool is_draw_cube = true;
      int count_point;
      float[,] generated_point;

      int size_grid = 60;
      int Step = 60;
      int Radius = 2;
      int half_picture_size = 300;
      double angleX;
```

```
        double angleY = Math.PI * 1.5d;
        double distance_z = 4d;
        int[] GAP_NUMS = new int[] { 1, 2, 3, 4, 5 };
        int[] start_mouse_pose;

        int[,,] density_3d;
        int[,] density;
        int[] density_up;
        int[] density_right;
        int max_dens_xy;
        int max_dens_x;
        int max_dens_y;
        int max_dens;

        public Form1()
        {
            InitializeComponent();
            OpenGL.MouseWheel += OpenGLMain_Control_MouseWheel;

            dataGridView1.RowHeadersVisible = false;
            dataGridView1.ColumnCount = 3;
            dataGridView1.Columns[0].Width = size_grid;
            dataGridView1.Columns[1].Width = size_grid;
            dataGridView1.Columns[2].Width = size_grid;
            dataGridView1.Width = size_grid * dataGridView1.ColumnCount + 20;
            Histograms.Image = new Bitmap(620, 620);
            Vertical_Histograms_pictureBox.Image = new Bitmap(220, 620);
            Horizontal_Histograms_pictureBox.Image = new Bitmap(620, 220);
            listBox1.SelectedIndex = 0;
        }
        public static double ConvertScale(double originalStart, double originalEnd, double newStart,
double newEnd, double value)
        {
            return newStart + ((value - originalStart) * ((double)(newEnd - newStart) / (originalEnd -
originalStart)));
        }
        private void Generate_button_Click(object sender, EventArgs e)
        {
```

```
            count_point = Int32.Parse(num_point.Text);
            is_generate_task = false;
            generate_var_but.Enabled = false;
            Start_but.Enabled = false;
            Open_but.Enabled = false;
            generate_points(is_task: is_generate_task, is_noise: is_noise_change);
        }
        public void generate_points(bool is_task = false, bool is_noise = false)
        {
            generated_point = new float[count_point, 3];
            float[] sum_generated_point = new float[3];
            for (int i = 0; i < count_point; i++)
            {
                if (!is_task) for (int j = 0; j < 3; j++) generated_point[i, j] = (float)Math.Round((double)random.Next(-100, 100) / 100, 2);
                else
                {
                    //x = 0.7 * cos(6 * pi * (i / count_point))
                    generated_point[i, 0] = check_point((float)(0.7f * Math.Cos(6d * Math.PI * (((double)i / count_point)))));
                    //y = 0.5 * sin(4 * pi * (i / count_point))
                    generated_point[i, 1] = check_point((float)(0.5f * Math.Sin(4d * Math.PI * ((double)i / count_point))));
                    //z = -1 + 2 * i / N
                    generated_point[i, 2] = check_point((float)(-1f + 2f * (float)i / count_point));
                }
                if (is_noise) for (int j = 0; j < 3; j++) sum_generated_point[j] += generated_point[i, j];
            }
            if (is_noise) for (int i = 0; i < count_point; i++) for (int j = 0; j < 3; j++) generated_point[i, j] = normal_distribution(generated_point[i, j], (float)(sum_generated_point[j] / count_point));

            Save_but.Enabled = true;
            picture_but.Enabled = true;
            view_datagreed();
            CalculateHistogramDensity();
            Draw3DGraph();
        }
        public float check_point(float x)
```

```
{
    if (x < -1f) return -1f;
    else if (x > 1f) return 1f;
    else return (float)Math.Round(x, 2);
}
public float normal_distribution(float a, float mean)
{
    double u = random.NextDouble();
    double v = random.NextDouble();
    float std_normal = (float)(Math.Sqrt(-2.0d * Math.Log(u)) * Math.Sin(2.0d * Math.PI * v));
    float gen_noise = (float)((float)(noise.Value / 1000) * std_normal) + a + mean;
    return gen_noise;
}
public void view_datagreed()
{
    if (count_point > 0)
    {
        dataGridView1.RowCount = 1;
        for (int i = 0; i < count_point; i++)
        {
            dataGridView1.RowCount += 1;
            dataGridView1.Rows[i].Cells[0].Value = Math.Round(generated_point[i, 0], 2);
            dataGridView1.Rows[i].Cells[1].Value = Math.Round(generated_point[i, 1], 2);
            dataGridView1.Rows[i].Cells[2].Value = Math.Round(generated_point[i, 2], 2);
        }
    }
}
private void Save_button_Click(object sender, EventArgs e)
{
    if (count_point != 0)
    {
        DialogResult res = saveFileDialog1.ShowDialog();
        try
        {
            if (res == DialogResult.OK) SaveCSV(saveFileDialog1.FileName);
        }
        catch
        {
```

```
            MessageBox.Show("Something wrong with your points");
        }
    }
}
public void SaveCSV(string csvPath)
{
    string data = "X;Y;Z\n";
    for (int i = 0; i < dataGridView1.RowCount - 1; i++)
    {
        data += dataGridView1.Rows[i].Cells[0].Value.ToString() + ";";
        data += dataGridView1.Rows[i].Cells[1].Value.ToString() + ";";
        data += dataGridView1.Rows[i].Cells[2].Value.ToString();
        data += "\n";
    }
    File.WriteAllText(csvPath, data);
}
public void OpenCSV(string csvPath)
{
    string csvContentStr = File.ReadAllText(csvPath);
    string[] vs = csvContentStr.Split('\n');
    string[] vs2;
    count_point = vs.Length - 2;
    num_point.Text = count_point.ToString();
    generated_point = new float[count_point, 3];
    for (int i = 1; i < count_point + 1; i++)
    {
        vs2 = vs[i].Split(';');
        for (int j = 0; j < 3; j++) generated_point[i - 1, j] = float.Parse(vs2[j]);
    }
    CalculateHistogramDensity();
    Draw3DGraph();
}
private void Download_button_Click(object sender, EventArgs e)
{
    try
    {
        DialogResult res = openFileDialog1.ShowDialog();
        if (res == DialogResult.OK)
```

```
            {
                OpenCSV(openFileDialog1.FileName);
                angleY_bar.Enabled = true;
                angleX_bar.Enabled = true;
                generate_var_but.Enabled = false;
                Save_but.Enabled = false;
                Refr_but.Enabled = true;
            }
            else MessageBox.Show("Error, you don't take any file.");
        }
        catch (Exception ex)
        {
            MessageBox.Show("Error, your file have incorrect type. You must take .csv.");
            MessageBox.Show(ex.Message);
        }
    }
    async void Draw3DGraph()
    {
        gl = null;
        gl = OpenGL.OpenGL;

        XYradioButton_CheckedChanged(new object(), new EventArgs());

        if (count_point == 0) return;
        gl.Clear(SharpGL.OpenGL.GL_COLOR_BUFFER_BIT                                    |
SharpGL.OpenGL.GL_DEPTH_BUFFER_BIT);
        gl.MatrixMode(SharpGL.OpenGL.GL_PROJECTION);
        gl.LoadIdentity();
        gl.Perspective(60.0f, OpenGL.Width / (double)OpenGL.Height, 0.01, 100.0);



        double CamX = distance_z * Math.Sin(angleY) * Math.Cos(angleX);
        double CamY = distance_z * Math.Cos(angleY);
        double CamZ = distance_z * Math.Sin(angleY) * Math.Sin(angleX);

        gl.LookAt(CamX, CamY, CamZ, 0, 0, 0, 0, 1, 0);

        gl.MatrixMode(SharpGL.OpenGL.GL_MODELVIEW);
```

```
    if (listBox1.SelectedIndex == 0)
    {
        draw_axes();
        draw_points();
        if (is_density.Checked)
            DrawFace();
        draw_cube_line();
    }
    else if (listBox1.SelectedIndex == 1)
    {
        draw_axes();
        DrawVoxels();
    }
    if (count_point != 0 && tabControl1.SelectedIndex == 1)
    {
        PaintingDensity();
        PaintHistograms();
    }

}
public void draw_cube_line()
{
    if (is_draw_cube) {
        gl.Begin(SharpGL.OpenGL.GL_LINES);
        gl.Color(1.0f, 1.0f, 1.0f);

        gl.Vertex(-1.0f, -1.0f, -1.0f);
        gl.Vertex(1.0f, -1.0f, -1.0f);

        gl.Vertex(-1.0f, -1.0f, -1.0f);
        gl.Vertex(-1.0f, 1.0f, -1.0f);

        gl.Vertex(-1.0f, -1.0f, -1.0f);
        gl.Vertex(-1.0f, -1.0f, 1.0f);

        gl.Vertex(1.0f, 1.0f, 1.0f);
        gl.Vertex(-1.0f, 1.0f, 1.0f);
```

```
            gl.Vertex(1.0f, 1.0f, 1.0f);
            gl.Vertex(1.0f, -1.0f, 1.0f);

            gl.Vertex(1.0f, 1.0f, 1.0f);
            gl.Vertex(1.0f, 1.0f, -1.0f);

            gl.Vertex(-1.0f, 1.0f, -1.0f);
            gl.Vertex(1.0f, 1.0f, -1.0f);

            gl.Vertex(-1.0f, 1.0f, -1.0f);
            gl.Vertex(-1.0f, 1.0f, 1.0f);

            gl.Vertex(1.0f, -1.0f, 1.0f);
            gl.Vertex(-1.0f, -1.0f, 1.0f);

            gl.Vertex(1.0f, -1.0f, -1.0f);
            gl.Vertex(1.0f, -1.0f, 1.0f);

            gl.Vertex(1.0f, 1.0f, -1.0f);
            gl.Vertex(1.0f, -1.0f, -1.0f);

            gl.Vertex(-1.0f, 1.0f, 1.0f);
            gl.Vertex(-1.0f, -1.0f, 1.0f);

            gl.End();
        }
    }
    public void draw_points()
    {
        gl.Begin(SharpGL.OpenGL.GL_POINTS);

        int Gap = 1;

        int GapCount = count_point;

        for (int i = 0; i < GapCount; i++)
            for (int j = 0; j < Gap; j++)
```

```
            if (GAP_NUMS.Contains(j + 1))
            {
                gl.Color((byte)255, (byte)255, (byte)255);
                gl.Vertex(generated_point[i * Gap + j, 0], generated_point[i * Gap + j, 1], generated_point[i * Gap + j, 2]);
            }

        gl.End();
    }
    public void draw_axes()
    {
        gl.Begin(SharpGL.OpenGL.GL_LINES);
        gl.Color(1.0f, 0.0f, 0.0f);
        gl.Vertex(0.0f, 0.0f, 0.0f);
        gl.Vertex(1.0f, 0.0f, 0.0f);

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

        gl.Color(0.0f, 0.0f, 1.0f);
        gl.Vertex(0.0f, 0.0f, 0.0f);
        gl.Vertex(0.0f, 0.0f, 1.0f);
        gl.End();
    }
    private void PointsControl_MouseDown(object sender, MouseEventArgs e)
    {
        is_rotate_change = true;
        start_mouse_pose = new int[] { e.X, e.Y };
    }
    private void PointsControl_MouseMove(object sender, MouseEventArgs e)
    {
        if (is_rotate_change)
        {
            angleX -= (float)((((double)(start_mouse_pose[0] - e.X)) * (Math.PI / 3)) / half_picture_size);
            angleY -= (float)((((double)(start_mouse_pose[1] - e.Y)) * (Math.PI / 6)) / half_picture_size);

            start_mouse_pose = new int[] { e.X, e.Y };
```

```
            try
            {
               angleX_bar.Value = check_angle((int)((180f / (float)Math.PI) * angleX));
               angleY_bar.Value = check_angle_y((int)((90f / (float)Math.PI) * angleY));
            }
            catch { }
            debug.Text = "Y angle is:" + Math.Round(angleY,2).ToString() + " X angle is:" +
Math.Round(angleX,2).ToString();
            Draw3DGraph();
         }
      }
      public int check_angle(int angle)
      {
         if (angle < -180 ) return -180;
         else if (angle > 180) return 180;
         return angle;
      }
      public int check_angle_y(int angle)
      {
         if (angle < 0) return 0;
         else if (angle > 180) return 180;
         return angle;
      }
      private void angleX_bar_ValueChanged(object sender, EventArgs e)
      {
         if (!is_rotate_change)
         {
            angleX = (float)(((float)Math.PI / 180f) * angleX_bar.Value);
            Draw3DGraph();
            debug.Text = "Y angle is:" + Math.Round(angleY, 2).ToString() + " X angle is:" +
Math.Round(angleX, 2).ToString();
         }
      }
      private void angleY_bar_ValueChanged(object sender, EventArgs e)
      {
         if (!is_rotate_change)
         {
            angleY = (float)(((float)Math.PI / 180f) * angleY_bar.Value) + Math.PI;
```

```
            Draw3DGraph();
            debug.Text = "Y angle is:" + Math.Round(angleY, 2).ToString() + " X angle is:" + Math.Round(angleX, 2).ToString();
        }
    }
    private void PointsControl_MouseUp(object sender, MouseEventArgs e)
    {
        is_rotate_change = false;
    }
    private void OpenGLMain_Control_MouseWheel(object sender, MouseEventArgs e)
    {
        if (e.Delta < 0) distance_z++;
        else distance_z--;
        Draw3DGraph();
    }
    public float id_to_coord(int d)
    {
        if (d < 5) return (float)((4 - d) * -0.2f) - 0.1f;
        return (d - 5) * 0.2f + 0.1f;
    }
    void DrawVoxels()
    {
        gl.Begin(SharpGL.OpenGL.GL_QUADS);
        for (int z = 0; z < 10; z ++)
        {
            for (int y = 0; y < 10; y++)
            {
                for (int x = 0; x < 10; x++)
                {
                    float size = (float)ConvertScale(0, max_dens, 0.01, 0.1, density_3d[x, y, z]);
                    if (size > 0.01f || !is_voxel_clear)
                    {
                        float fX = id_to_coord(x);
                        float fY = id_to_coord(y);
                        float fZ = id_to_coord(z);

                        // Front edge
                        gl.Color(0.25f, 0.25f, 0.25f);
```

```
gl.Vertex(fX - size, fY + size, fZ + size);
gl.Vertex(fX + size, fY + size, fZ + size);
gl.Vertex(fX + size, fY - size, fZ + size);
gl.Vertex(fX - size, fY - size, fZ + size);

// Back edge
gl.Color(0.5f, 0.5f, 0.5f);
gl.Vertex(fX - size, fY + size, fZ - size);
gl.Vertex(fX + size, fY + size, fZ - size);
gl.Vertex(fX + size, fY - size, fZ - size);
gl.Vertex(fX - size, fY - size, fZ - size);

// The Bottom edge
gl.Color(0.35f, 0.35f, 0.35f);
gl.Vertex(fX + size, fY - size, fZ - size);
gl.Vertex(fX + size, fY - size, fZ + size);
gl.Vertex(fX - size, fY - size, fZ + size);
gl.Vertex(fX - size, fY - size, fZ - size);

// The Upper edge
gl.Color(0.85f, 0.85f, 0.85f);
gl.Vertex(fX + size, fY + size, fZ - size);
gl.Vertex(fX + size, fY + size, fZ + size);
gl.Vertex(fX - size, fY + size, fZ + size);
gl.Vertex(fX - size, fY + size, fZ - size);

// Left edge
gl.Color(1.0f, 1.0f, 1.0f);
gl.Vertex(fX - size, fY + size, fZ - size);
gl.Vertex(fX - size, fY + size, fZ + size);
gl.Vertex(fX - size, fY - size, fZ + size);
gl.Vertex(fX - size, fY - size, fZ - size);

// Right edge
gl.Color(0.2f, 0.2f, 0.2f);
gl.Vertex(fX + size, fY + size, fZ - size);
gl.Vertex(fX + size, fY + size, fZ + size);
gl.Vertex(fX + size, fY - size, fZ + size);
```

```
                gl.Vertex(fX + size, fY - size, fZ - size);
            }
        }
    }
    gl.End();
}
private void DrawFace()
{
    var Nurb = gl.NewNurbsRenderer();
    float[] Knots = { 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f };

    for (int y = 0; y < 10; y++)
        for (int x = 0; x < 10; x++)
        {
            float Height = density[x, y] / 100f;
            float Hue = (float)ConvertScale(0d, 0.56d, 0.1d, 1d, (double)Height);
            Height -= 1.0f;
            float[] PointsXY = {
                -0.4f*2 + x * 0.2f, -0.4f*2 + y * 0.2f, -1f,
                -0.4f*2 + x * 0.2f, -0.43f*2 + y * 0.2f, -1f,
                -0.4f*2 + x * 0.2f, -0.47f*2 + y * 0.2f, -1f,
                -0.4f*2 + x * 0.2f, -0.5f*2 + y * 0.2f, -1f,

                -0.43f*2 + x * 0.2f, -0.4f*2 + y * 0.2f, -1f,
                -0.43f*2 + x * 0.2f, -0.43f*2 + y * 0.2f, Height,
                -0.43f*2 + x * 0.2f, -0.47f*2 + y * 0.2f, Height,
                -0.43f*2 + x * 0.2f, -0.5f*2 + y * 0.2f, -1f,

                -0.47f*2 + x * 0.2f, -0.4f*2 + y * 0.2f, -1f,
                -0.47f*2 + x * 0.2f, -0.43f*2 + y * 0.2f, Height,
                -0.47f*2 + x * 0.2f, -0.47f*2 + y * 0.2f, Height,
                -0.47f*2 + x * 0.2f, -0.5f*2 + y * 0.2f, -1f,

                -0.5f*2 + x * 0.2f,  -0.4f*2 + y * 0.2f, -1f,
                -0.5f*2 + x * 0.2f, -0.43f*2 + y * 0.2f, -1f,
                -0.5f*2 + x * 0.2f, -0.47f*2 + y * 0.2f, -1f,
                -0.5f*2 + x * 0.2f, -0.5f*2 + y * 0.2f, -1f,
```

```
                };

                float[] Points = null;
                gl.Color(Hue, 0, Hue);
                Points = PointsXY;
                gl.BeginSurface(Nurb);
                gl.NurbsSurface(Nurb, Knots.Length, Knots, Knots.Length, Knots, 4 * 3, 3, Points, 4, 4,
SharpGL.OpenGL.GL_MAP2_VERTEX_3);
                gl.EndSurface(Nurb);
            }
        }
        private void CalculateHistogramDensity()
        {
            max_dens_xy = 0;
            max_dens_x = 0;
            max_dens_y = 0;
            max_dens = 0;
            density_up = new int[10];
            density_right = new int[10];
            density = new int[10, 10];
            density_3d = new int[10, 10, 10];
            for (int i = 0; i < count_point; i++)
            {
                int id_x = check_id_density(generated_point[i, 0]);
                int id_y = check_id_density(generated_point[i, 1]);
                int id_z = check_id_density(generated_point[i, 2]);
                density_up[id_x]++;
                density_right[id_y]++;
                density[id_x, id_y]++;
                density_3d[id_x, id_y, id_z]++;
                if (density[id_x, id_y] > max_dens_xy) max_dens_xy = density[id_x, id_y];
                if (density_up[id_x] > max_dens_y) max_dens_y = density_up[id_x];
                if (density_right[id_y] > max_dens_x) max_dens_x = density_right[id_y];
                if (density_3d[id_x, id_y, id_z] > max_dens) max_dens = density_3d[id_x, id_y, id_z];
            }

        }
        public int check_id_density(float d)
```

```
    {
        if (d == 0f)                                    return 4;
        else if (d == -1f || 4 + (int)Math.Ceiling(d * 100 / 20) <= 0) return 0;
        else if (d < 0f)                                return 4 + (int)Math.Ceiling(d * 100 / 20);
        else if (d == 1f || 4 + (int)Math.Ceiling(d * 100 / 20) >= 9)  return 9;
        else                                            return 4 + (int)Math.Ceiling(d * 100 / 20);
    }
    public Brush create_color_dens(int f)
    {
        int ff = (int)(f * 255 / max_dens_xy);
        if (ff == 0) ff = 10;
        return new SolidBrush(System.Drawing.Color.FromArgb(ff, 0, 0, 0));
    }
    private void PaintingDensity()
    {
        Graphics g = Graphics.FromImage(Histograms.Image);
        g.FillRectangle(Brushes.White, new Rectangle(0, 0, Histograms.Image.Width,
Histograms.Image.Height));
        for (int y = 0; y < 10; y++)
            for (int x = 0; x < 10; x++)
            {
                Brush brush = create_color_dens(density[y, x]);
                g.FillRectangle(brush, 10 + Step * y, 10 + (Step * x), Step, Step);
            }
        for (int i = 0; i < count_point; i++)
        {
            float X = (float)(ConvertScale(-1, 1, 10, 610, generated_point[i, 0]) - Radius);
            float Y = (float)(ConvertScale(1, -1, 10, 610, generated_point[i, 1]) - Radius);
            g.FillEllipse(new SolidBrush(Color.Blue), X, Y, Radius * 2, Radius * 2);
        }
        Histograms.Refresh();
    }
    private void PaintHistograms()
    {
        Graphics g = Graphics.FromImage(Horizontal_Histograms_pictureBox.Image);
        g.FillRectangle(Brushes.White, new Rectangle(0, 0,
Horizontal_Histograms_pictureBox.Image.Width, Horizontal_Histograms_pictureBox.Image.Height));
        Graphics g2 = Graphics.FromImage(Vertical_Histograms_pictureBox.Image);
```

```
            g2.FillRectangle(Brushes.White, new Rectangle(0, 0, Vertical_Histograms_pictureBox.Image.Width, Vertical_Histograms_pictureBox.Image.Height));
            int MaxX = max_in_array(density_right);
            int MaxY = max_in_array(density_up);

            for (int i = 0; i < 10; i++)
            {

                g.FillRectangle(Brushes.Black,  10 + Step * i + 1, 210 - (int)((density_up[i] * 200) / MaxY), Step - 1, (int)((density_up[i] * 200) / MaxY));
                g2.FillRectangle(Brushes.Black, 10, 10 + Step * i + 1, (int)((density_right[i] * 200) / MaxX), Step - 1);
            }
            Horizontal_Histograms_pictureBox.Refresh();
            Vertical_Histograms_pictureBox.Refresh();
        }
        public int max_in_array(int[] ff)
        {
            int max = 0;
            for (int i = 0; i < 10; i++)
            {
                if (ff[i] > max) max = ff[i];
            }
            return max;
        }
        private void XYradioButton_CheckedChanged(object sender, EventArgs e)
        {
            if (count_point != 0 && tabControl1.SelectedIndex == 1)
            {
                PaintingDensity();
                PaintHistograms();
            }
            else return;
        }
        private void tabControl1_SelectedIndexChanged(object sender, EventArgs e)
        {
            XYradioButton_CheckedChanged(sender, e);
        }
```

```
        private void Surface_checkBox_CheckedChanged(object sender, EventArgs e)
        {
            Draw3DGraph();
        }
        private Mat SaveScreen(OpenGLControl control)
        {
            OpenGL gl = control.OpenGL;

            int h = control.Height;
            int w = control.Width;
            byte[] Pixels = new byte[4 * w * h];
            gl.ReadPixels(0, 0, w, h, SharpGL.OpenGL.GL_BGRA,
SharpGL.OpenGL.GL_UNSIGNED_BYTE, Pixels);

            Mat Screenshot = new Mat(h, w, MatType.CV_8UC4);
            Marshal.Copy(Pixels, 0, Screenshot.Data, 4 * w * h);
            Cv2.Flip(Screenshot, Screenshot, FlipMode.X);

            return Screenshot;
        }
        private void SaveImage_button_Click(object sender, EventArgs e)
        {
            if (OpenGL == null) return;
            DialogResult res = saveFileDialog1.ShowDialog();
            try
            {
                if (res == DialogResult.OK)
                {
                    listBox1.SelectedIndex = 0;
                    Draw3DGraph();
                    Mat PointsScreenshot = SaveScreen(OpenGL);
                    listBox1.SelectedIndex = 1;
                    Draw3DGraph();
                    Mat VoxelsScreenshot = SaveScreen(OpenGL);

                    Mat FinalScreenshot = new Mat();
                    Cv2.HConcat(new Mat[] { PointsScreenshot, VoxelsScreenshot }, FinalScreenshot);
                    FinalScreenshot.SaveImage(saveFileDialog1.FileName);
```

```
            }
        }
        catch
        {
            MessageBox.Show("Something wrong with your picture");
        }
    }
    private void noise_ValueChanged(object sender, EventArgs e)
    {
        count_point = Int32.Parse(num_point.Text);
        if (noise.Value > 0) is_noise_change = true;
        else is_noise_change = false;
        generate_points(is_task: is_generate_task, is_noise: is_noise_change);
    }
    private void generate_var_but_Click(object sender, EventArgs e)
    {
        count_point = Int32.Parse(num_point.Text);
        is_generate_task = true;
        generate_var_but.Enabled = false;
        Start_but.Enabled = false;
        Open_but.Enabled = false;
        generate_points(is_task: is_generate_task, is_noise: is_noise_change);
    }
    private void listBox1_Click(object sender, EventArgs e)
    {
        if (listBox1.SelectedIndex == 1)
        {
            is_density.Enabled = false;
            is_density.Checked = false;
        }
        else is_density.Enabled = true;
        Draw3DGraph();
    }

    private void Refr_but_Click(object sender, EventArgs e)
    {
        dataGridView1.Rows.Clear();
        dataGridView1.Refresh();
```

```
            angleX = 0d;
            angleY = Math.PI * 1.5d;
            distance_z = 4d;
            angleY_bar.Enabled = false;
            angleX_bar.Enabled = false;
            generate_var_but.Enabled = true;
            Start_but.Enabled = true;
            Open_but.Enabled = true;
            Save_but.Enabled = false;
            picture_but.Enabled = false;
            noise.Value = 0;
            gl.Clear(SharpGL.OpenGL.GL_COLOR_BUFFER_BIT                                    |
SharpGL.OpenGL.GL_DEPTH_BUFFER_BIT);
        }
    }
}
```