МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



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) 10х10 ячеек в координатной плоскости XY.

# 

Рисунок 5 - Отображение гистограмм

# Выполнить статистический анализ набора точек, выполнив расчет плотности заполнения вокселей (voxel) как пространственной матрицы 10х10х10 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

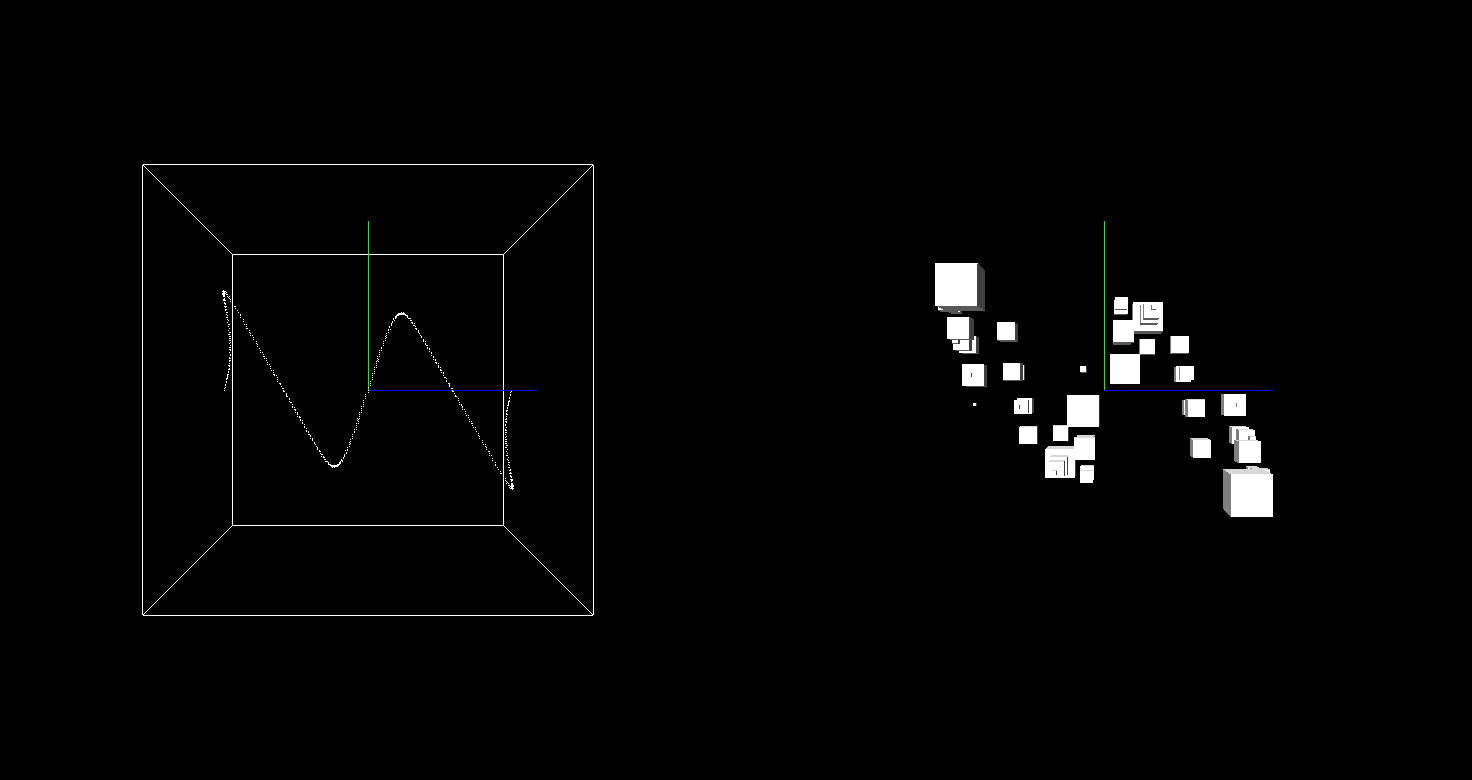


Рисунок 10 - Сохраненное изображение

# 

Рисунок 11 - Финальный вид программы

**Вывод**

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

# Приложение A

Листинг A-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);  }  }  } |