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

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»



Кафедра СМАРТ технологий

Лабораторная работа №4

«Многомерный анализ и визуализация мониторинговых 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 визуализацию сцены для отображения пространственных данных в виде куба с окружающими поверхностями для отображения графиков. Реализовать управление отображением с помощью мыши с центром вращения сцены в геометрическом центре куба.

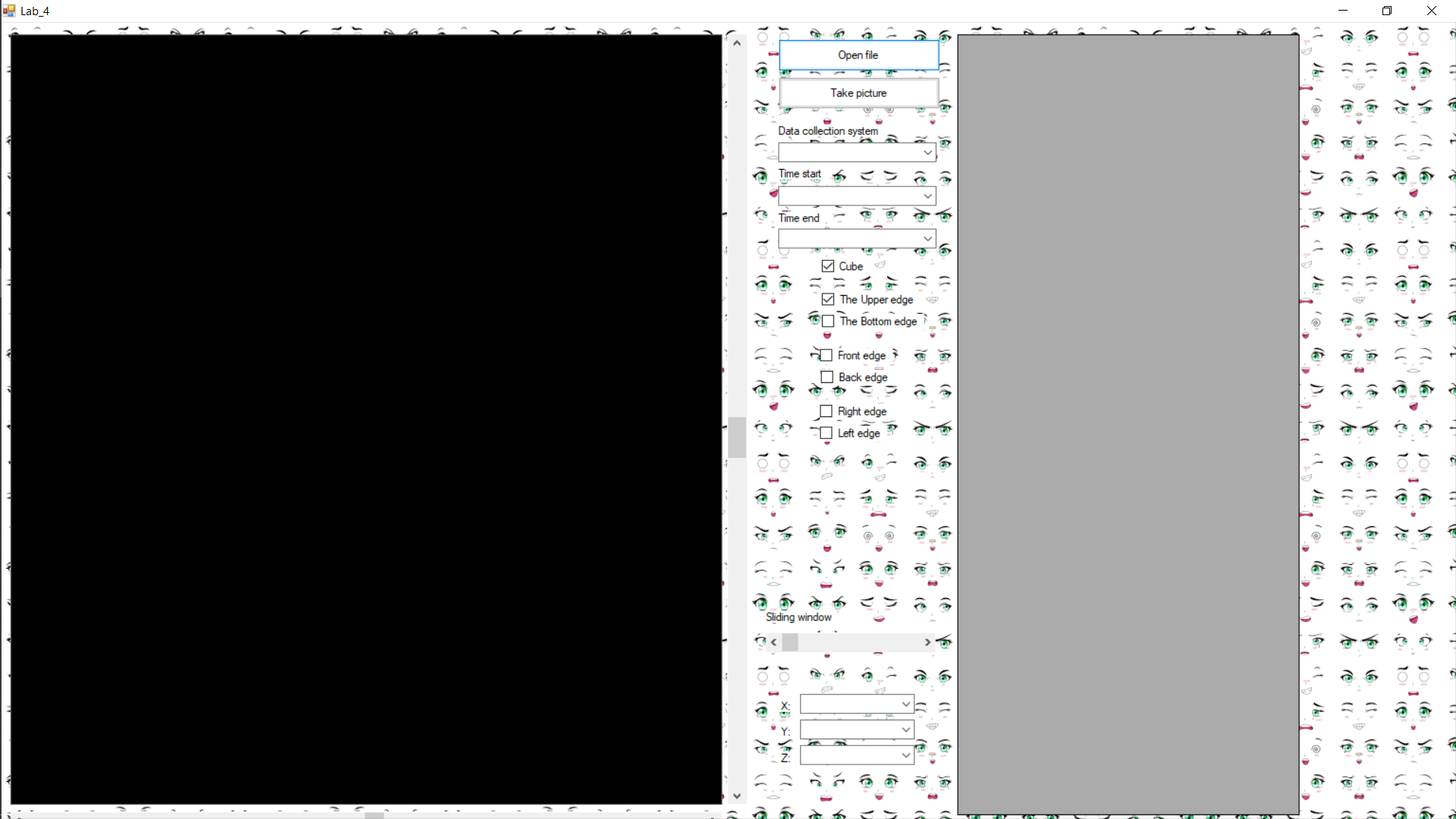


Рисунок – Form

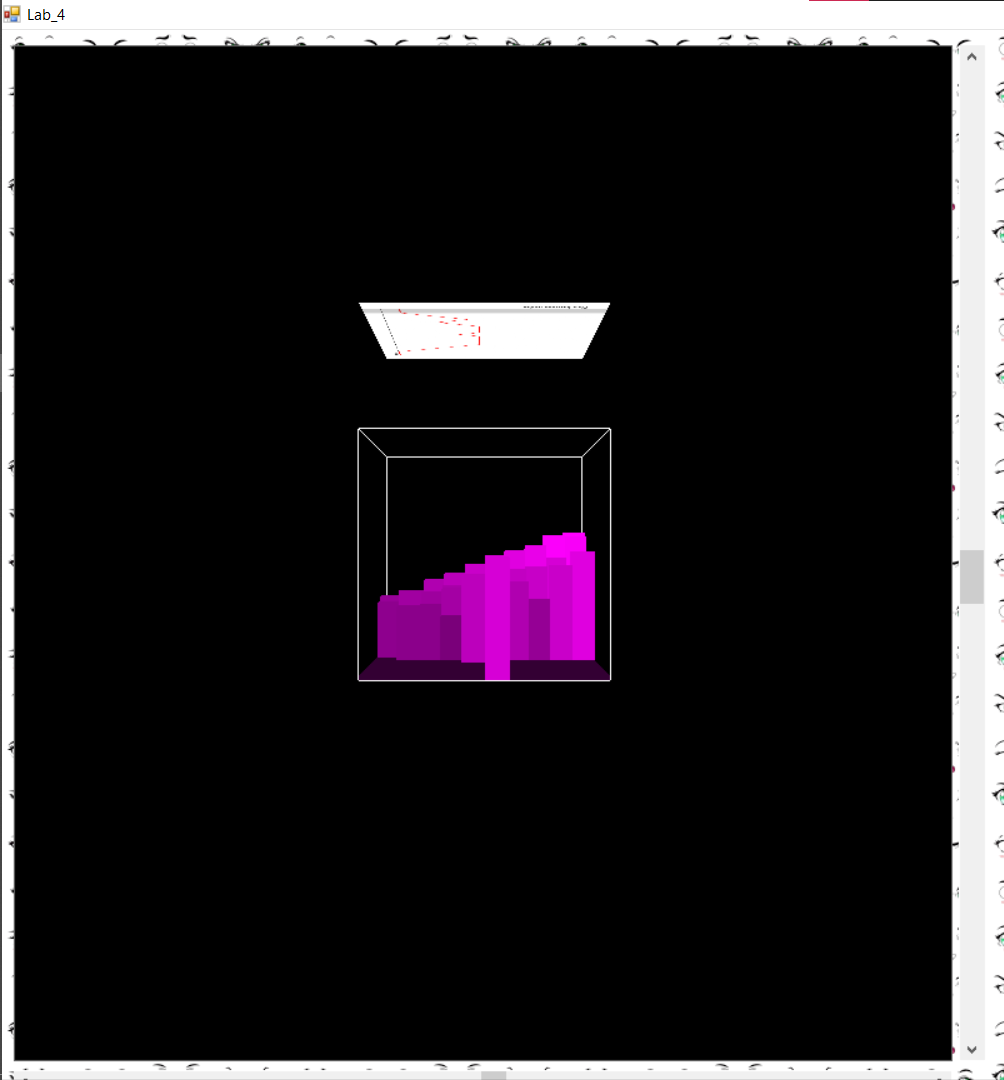


Рисунок - Отображение данных в 3д

* Реализовать возможность скрывать куб или отдельные поверхности для графиков.

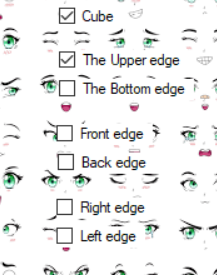


Рисунок - Переключение отображений

* Реализовать загрузку набора данных, представленных как набор мониторинговых параметров из сообщений в сети CAN. Выполнить фильтрацию сообщений по заданной карте и квантование по моменту сбора данных.

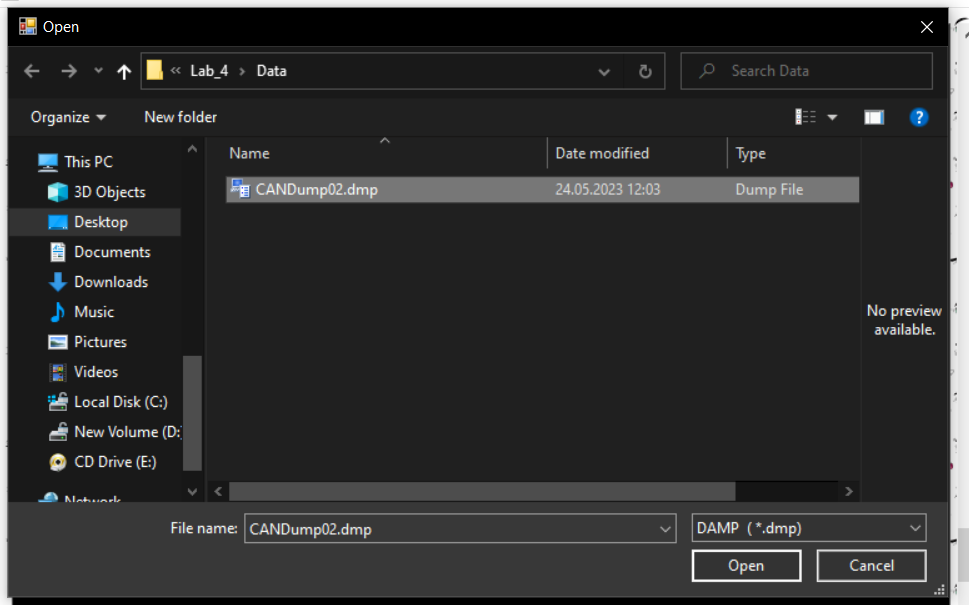
****

Рисунок - Открытие файла с данными

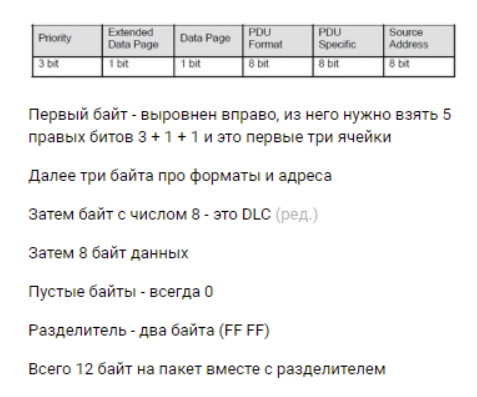


Рисунок - Формат данных в файле

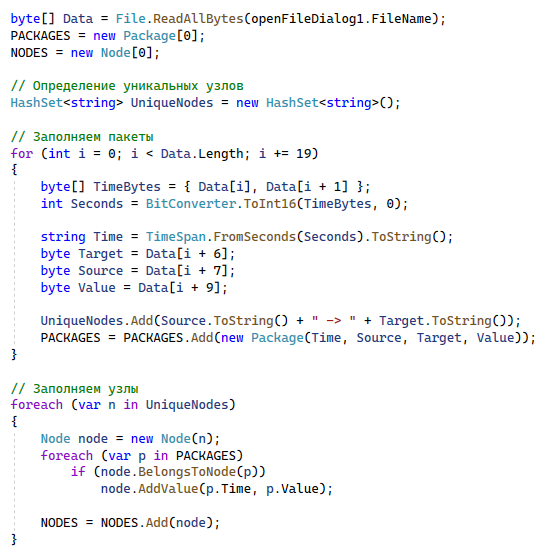


Рисунок - Обработка байтов из файла и распределение по пакетам

* Выполнить статистический анализ записей как набора точек в многомерном пространстве, вычисляя параметры описательной статистики методом скользящих сводных параметров. Реализовать возможность задать глубину истории рассчитываемых параметров. еализовать систему визуализации данных (плоскости и графиков) в режиме скользящих сводных параметров.

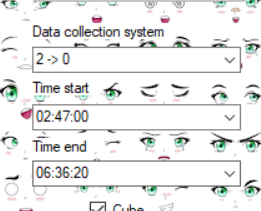
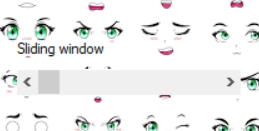


Рисунок - Элементы управления

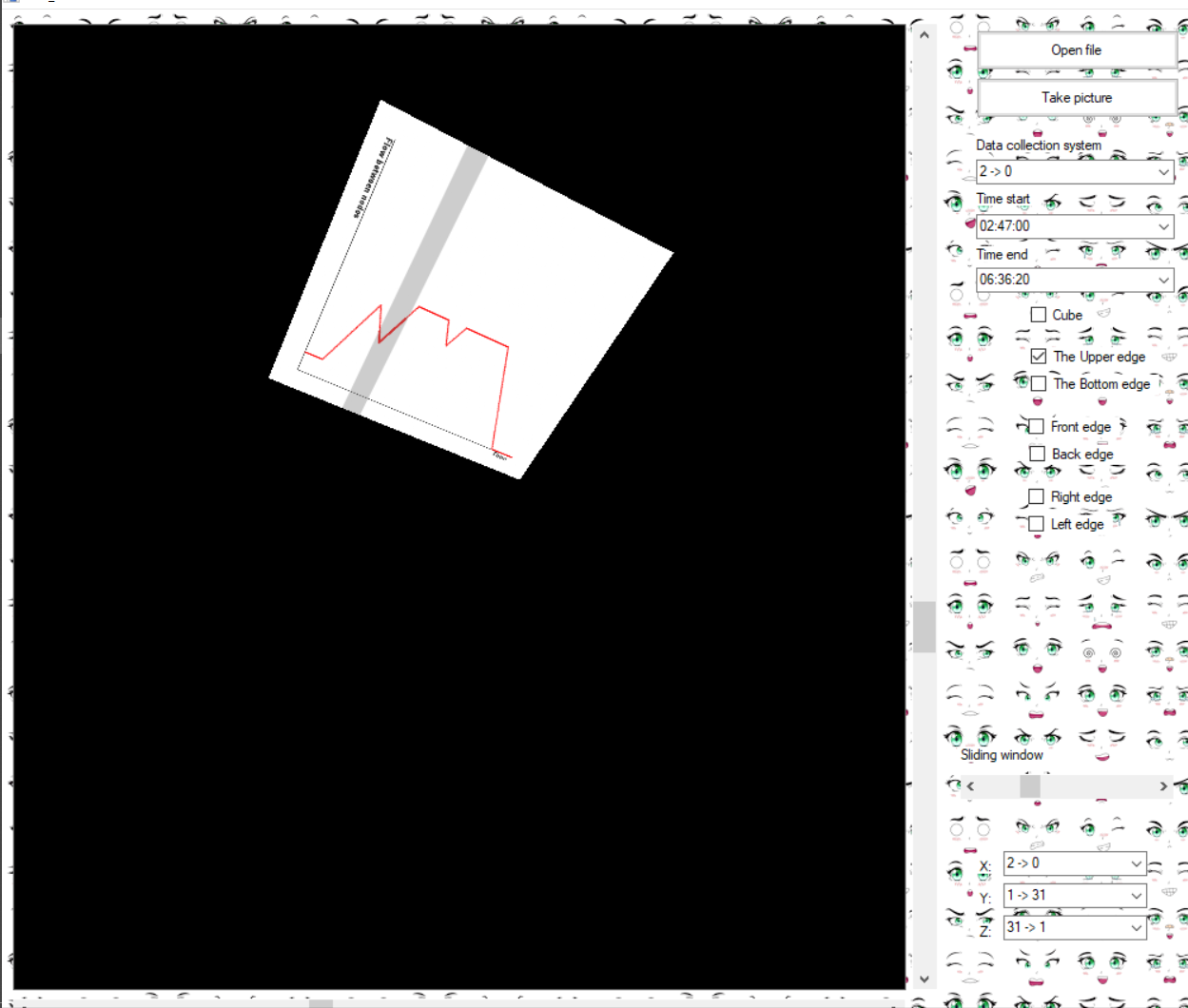


Рисунок - Отображенние текстуры Bitmap

* Реализовать сохранение полученного изображения в файл

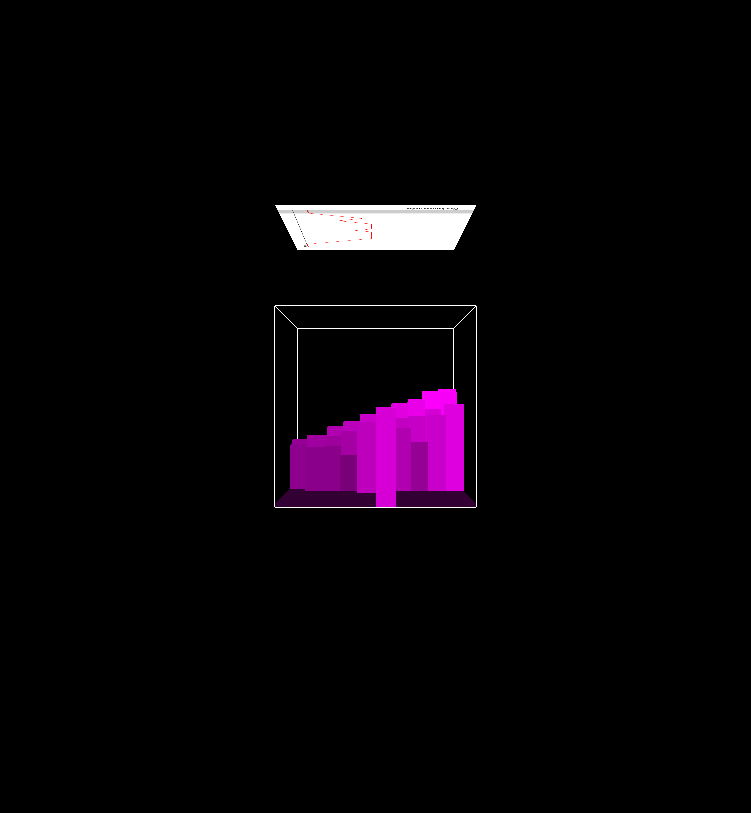


Рисунок - Пример изображения.

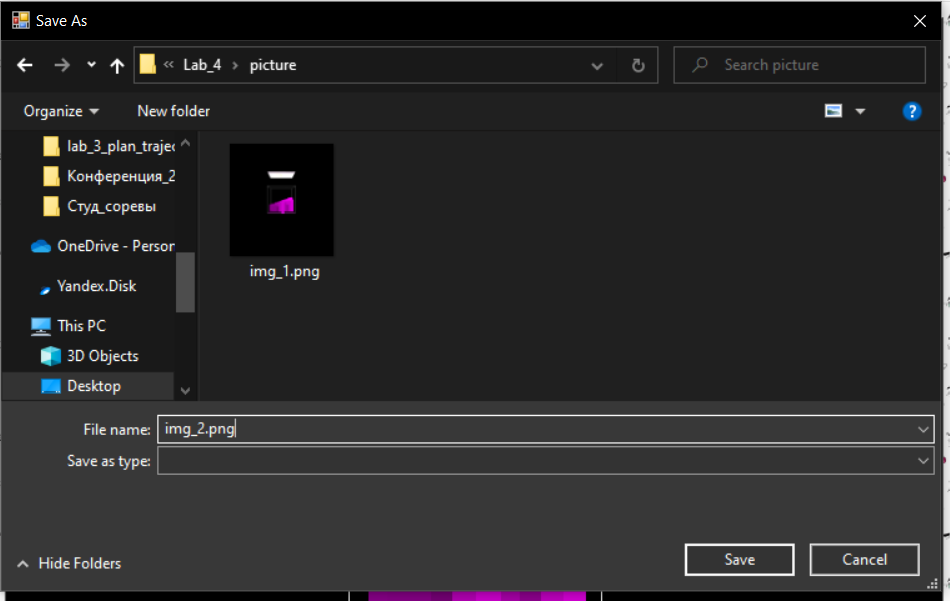


Рисунок - Сохранение изображения

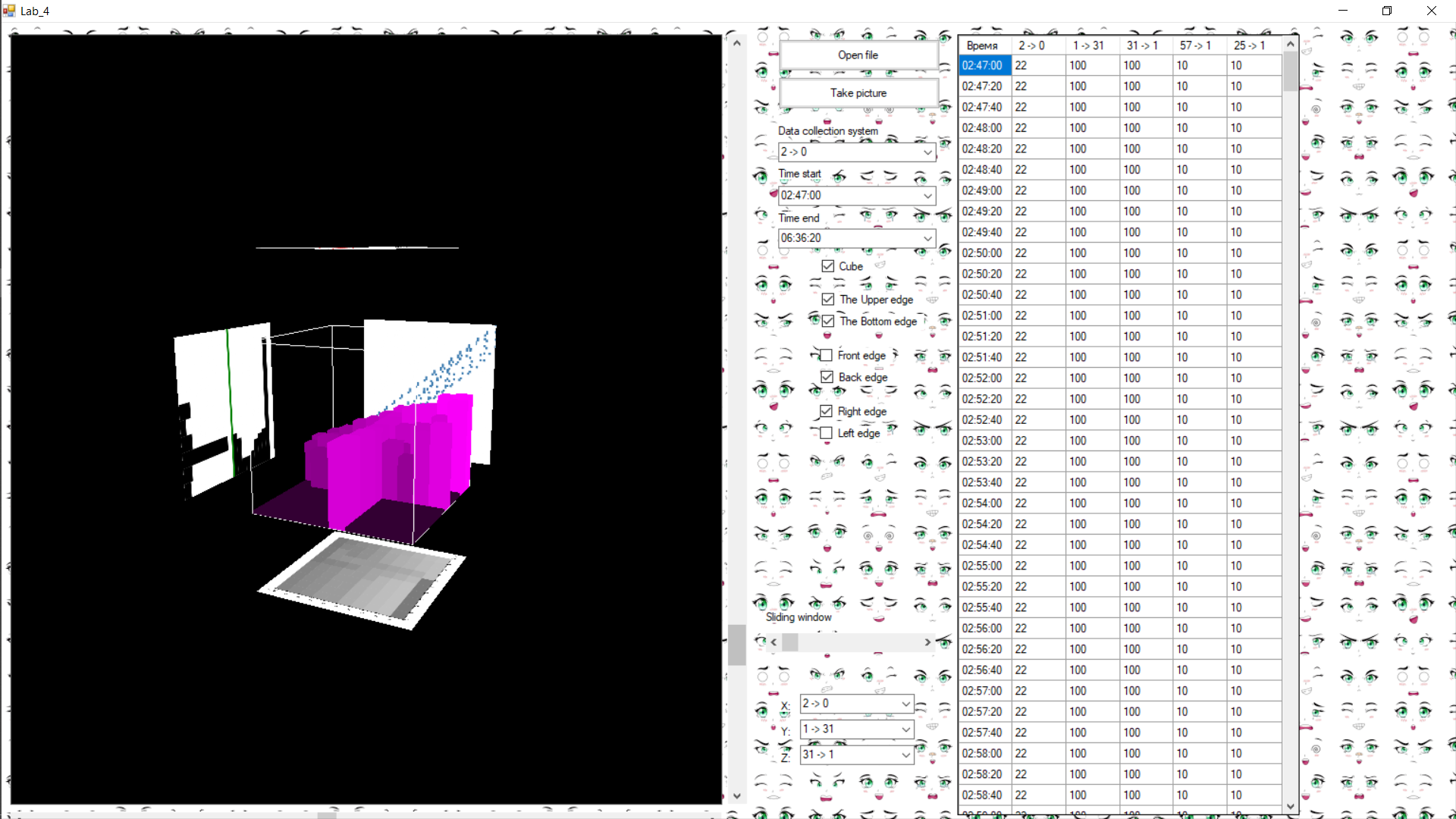


Рисунок - Итоговый вид рабочей программы

**Вывод**

В ходе создания программы было написанно ПО для отображения данных из файла .dmp с их анализом.

# Приложение A

Листинг A-1 – программный код:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Windows.Forms;  using SharpGL;  using OpenCvSharp;  using System.Runtime.InteropServices;  using SharpGL.SceneGraph.Assets;  using System.IO;  using System.Drawing.Drawing2D;  using System.Drawing.Text;  using System.Threading.Tasks;  namespace Lab\_4  {  public partial class Form1 : Form  {  private bool ROTATING;  bool is\_rotate\_change = false;  private int START\_X;  private int START\_Y;  private double PREV\_EQUATOR;  private double PREV\_MERIDIAN;  double angleX;  double angleY = Math.PI \* 1.5d;  double distance\_z = 40d;  int[] start\_mouse\_pose;  int half\_picture\_size = 500;  private Package[] PACKAGES;  private Node[] NODES;  private int[,] Z\_HEIGHTS;  private int[,] Z\_VALUES;  private const string RIGHT\_PATH = "Right.jpg";  private const string FRONT\_PATH = "Front.jpg";  private const string TOP\_PATH = "Top.jpg";  private const string BOTTOM\_PATH = "Bottom.jpg";  private const string LEFT\_PATH = "Left.jpg";  private const string BACK\_PATH = "Back.jpg";  OpenGL opengl1;  public Form1()  {  InitializeComponent();  DataControl.MouseWheel += DataControl\_MouseWheel;  DeletePlanes();  opengl1 = DataControl.OpenGL;  }  private void DeletePlanes()  {  if (File.Exists(RIGHT\_PATH)) File.Delete(RIGHT\_PATH);  if (File.Exists(FRONT\_PATH)) File.Delete(FRONT\_PATH);  if (File.Exists(TOP\_PATH)) File.Delete(TOP\_PATH);  if (File.Exists(BOTTOM\_PATH)) File.Delete(BOTTOM\_PATH);  }  private void FillDataGrid(HashSet<string> headers)  {  CANDataGrid.Rows.Clear();  string[] HeaderNames = new string[1];  HeaderNames[0] = "Время";  foreach (var h in headers)  HeaderNames = HeaderNames.Add(h);  foreach (var name in HeaderNames)  {  DataGridViewColumn Column = new DataGridViewColumn  {  CellTemplate = new DataGridViewTextBoxCell(),  HeaderText = name  };  CANDataGrid.Columns.Add(Column);  }  int TimeCount = NODES[0].TimeStamps.Length;  for (int t = 0; t < TimeCount; t++)  {  string[] RowData = new string[NODES.Length + 1];  RowData[0] = NODES[0].TimeStamps[t];  for (int n = 0; n < NODES.Length; n++)  RowData[n + 1] = NODES[n].Values[t].ToString();  CANDataGrid.Rows.Add(RowData);  }  }  private Mat GetScreenshot(OpenGLControl control)  {  OpenGL opengl1 = control.OpenGL;  int h = control.Height;  int w = control.Width;  byte[] Pixels = new byte[4 \* w \* h];  opengl1.ReadPixels(0, 0, w, h, OpenGL.GL\_BGRA, 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 async void DrawEverything()  {  try  {  opengl1.Clear(OpenGL.GL\_COLOR\_BUFFER\_BIT | OpenGL.GL\_DEPTH\_BUFFER\_BIT);  opengl1.MatrixMode(OpenGL.GL\_PROJECTION);  opengl1.LoadIdentity();  opengl1.Perspective(60.0f, DataControl.Width / (double)DataControl.Height, 0.01, 100.0);  short ScaleX = 1;  short ScaleY = 1;  short ScaleZ = 1;  opengl1.Scale(ScaleX, ScaleY, ScaleZ);  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);  opengl1.LookAt(CamX, CamY, CamZ, 0, 0, 0, 0, 1, 0);  opengl1.MatrixMode(OpenGL.GL\_MODELVIEW);  const float shift = 5.0f;  const float size = 10.0f;  float qx = size / -2.0f;  float qy = size / -2.0f;  float qz = size / -2.0f;  opengl1.Color(1.0f, 1.0f, 1.0f);  // Задание 1  if (HideCubeCheckBox.Checked)  {  if (PACKAGES == null) return;  // Куб  await DrawCube(size);  // Задание 4  await ThreeParamsDependency();  }  opengl1.Color(1.0f, 1.0f, 1.0f);  opengl1.Enable(OpenGL.GL\_TEXTURE\_2D);  //------------------------------------------------------------------Правая поверхность  Texture RightTexture;  if (File.Exists(RIGHT\_PATH))  {  RightTexture = new Texture();  RightTexture.Create(opengl1, RIGHT\_PATH);  RightTexture.Bind(opengl1);  }  opengl1.Begin(OpenGL.GL\_QUADS);  if (HideRightPlaneCheckBox.Checked)  {  await DrawRightPanel(qx, qy, qz, size, shift);  }  opengl1.End();  //------------------------------------------------------------------Передняя поверхность  Texture FrontTexture;  if (File.Exists(FRONT\_PATH))  {  FrontTexture = new Texture();  FrontTexture.Create(opengl1, FRONT\_PATH);  FrontTexture.Bind(opengl1);  }  opengl1.Begin(OpenGL.GL\_QUADS);  if (HideFrontPlaneCheckBox.Checked)  {  await DrawFrontPanel(qx, qy, qz, size, shift);  }  opengl1.End();  //------------------------------------------------------------------Верхняя поверхность  Texture TopTexture;  if (File.Exists(TOP\_PATH))  {  TopTexture = new Texture();  TopTexture.Create(opengl1, TOP\_PATH);  TopTexture.Bind(opengl1);  }  opengl1.Begin(OpenGL.GL\_QUADS);  if (HideTopPlaneCheckBox.Checked)  {  await DrawTopPanel(qx, qy, qz, size, shift);  }  opengl1.End();  //------------------------------------------------------------------Левая поверхность  Texture LeftTexture;  if (File.Exists(LEFT\_PATH))  {  LeftTexture = new Texture();  LeftTexture.Create(opengl1, LEFT\_PATH);  LeftTexture.Bind(opengl1);  }  opengl1.Begin(OpenGL.GL\_QUADS);  if (HideLeftPlaneCheckBox.Checked)  await DrawLeftPanel(qx, qy, qz, size, shift);  opengl1.End();  //------------------------------------------------------------------Задняя поверхность  Texture BackTexture;  if (File.Exists(BACK\_PATH))  {  BackTexture = new Texture();  BackTexture.Create(opengl1, BACK\_PATH);  BackTexture.Bind(opengl1);  }  opengl1.Begin(OpenGL.GL\_QUADS);  if (HideBackPlaneCheckBox.Checked)  {  await DrawBackPanel(qx, qy, qz, size, shift);  }  opengl1.End();  //------------------------------------------------------------------Нижняя поверхность  Texture BottomTexture;  if (File.Exists(BOTTOM\_PATH))  {  BottomTexture = new Texture();  BottomTexture.Create(opengl1, BOTTOM\_PATH);  BottomTexture.Bind(opengl1);  }  opengl1.Begin(OpenGL.GL\_QUADS);  if (HideBottomPlaneCheckBox.Checked)  {  await DrawBottomPanel(qx, qy, qz, size, shift);  }  opengl1.End();  opengl1.Flush();  }  catch { }  }  Task DrawRightPanel(float qx, float qy, float qz, float size, float shift)  {  opengl1.TexCoord(0, 0);  opengl1.Vertex(qx + size + shift, qy, qz);  opengl1.TexCoord(0, 1);  opengl1.Vertex(qx + size + shift, qy, qz + size);  opengl1.TexCoord(1, 1);  opengl1.Vertex(qx + size + shift, qy + size, qz + size);  opengl1.TexCoord(1, 0);  opengl1.Vertex(qx + size + shift, qy + size, qz);  return Task.CompletedTask;  }  Task DrawTopPanel(float qx, float qy, float qz, float size, float shift)  {  opengl1.TexCoord(0, 1);  opengl1.Vertex(qx, qy + size + shift, qz);  opengl1.TexCoord(0, 0);  opengl1.Vertex(qx, qy + size + shift, qz + size);  opengl1.TexCoord(1, 0);  opengl1.Vertex(qx + size, qy + size + shift, qz + size);  opengl1.TexCoord(1, 1);  opengl1.Vertex(qx + size, qy + size + shift, qz);  return Task.CompletedTask;  }  Task DrawBottomPanel(float qx, float qy, float qz, float size, float shift)  {  opengl1.TexCoord(0, 0);  opengl1.Vertex(qx, qy - shift, qz);  opengl1.TexCoord(0, 1);  opengl1.Vertex(qx, qy - shift, qz + size);  opengl1.TexCoord(1, 1);  opengl1.Vertex(qx + size, qy - shift, qz + size);  opengl1.TexCoord(1, 0);  opengl1.Vertex(qx + size, qy - shift, qz);  return Task.CompletedTask;  }  Task DrawFrontPanel(float qx, float qy, float qz, float size, float shift)  {  opengl1.TexCoord(0, 0);  opengl1.Vertex(qx, qy, qz + size + shift);  opengl1.TexCoord(1, 0);  opengl1.Vertex(qx + size, qy, qz + size + shift);  opengl1.TexCoord(1, 1);  opengl1.Vertex(qx + size, qy + size, qz + size + shift);  opengl1.TexCoord(0, 1);  opengl1.Vertex(qx, qy + size, qz + size + shift);  return Task.CompletedTask;  }  Task DrawBackPanel(float qx, float qy, float qz, float size, float shift)  {  opengl1.TexCoord(0, 0);  opengl1.Vertex(qx, qy, qz - shift);  opengl1.TexCoord(1, 0);  opengl1.Vertex(qx + size, qy, qz - shift);  opengl1.TexCoord(1, 1);  opengl1.Vertex(qx + size, qy + size, qz - shift);  opengl1.TexCoord(0, 1);  opengl1.Vertex(qx, qy + size, qz - shift);  return Task.CompletedTask;  }  Task DrawLeftPanel(float qx, float qy, float qz, float size, float shift)  {  opengl1.TexCoord(0, 0);  opengl1.Vertex(qx - shift, qy, qz);  opengl1.TexCoord(1, 0);  opengl1.Vertex(qx - shift, qy, qz + size);  opengl1.TexCoord(1, 1);  opengl1.Vertex(qx - shift, qy + size, qz + size);  opengl1.TexCoord(0, 1);  opengl1.Vertex(qx - shift, qy + size, qz);  return Task.CompletedTask;  }  private Task DrawCube(float size)  {  float x = size / -2.0f;  float y = size / -2.0f;  float z = size / -2.0f;  opengl1.Begin(OpenGL.GL\_LINES);  opengl1.Vertex(x, y, z); opengl1.Vertex(x, y, z + size);  opengl1.Vertex(x, y, z + size); opengl1.Vertex(x, y + size, z + size);  opengl1.Vertex(x, y + size, z + size); opengl1.Vertex(x, y + size, z);  opengl1.Vertex(x, y + size, z); opengl1.Vertex(x, y, z);  opengl1.Vertex(x, y, z); opengl1.Vertex(x + size, y, z);  opengl1.Vertex(x + size, y, z); opengl1.Vertex(x + size, y + size, z);  opengl1.Vertex(x + size, y + size, z); opengl1.Vertex(x, y + size, z);  opengl1.Vertex(x, y + size, z); opengl1.Vertex(x, y, z);  opengl1.Vertex(x + size, y, z + size); opengl1.Vertex(x + size, y, z);  opengl1.Vertex(x + size, y, z + size); opengl1.Vertex(x + size, y + size, z + size);  opengl1.Vertex(x + size, y, z + size); opengl1.Vertex(x, y, z + size);  opengl1.Vertex(x + size, y + size, z + size); opengl1.Vertex(x + size, y + size, z);  opengl1.Vertex(x + size, y + size, z + size); opengl1.Vertex(x, y + size, z + size);  opengl1.End();  return Task.CompletedTask;  }  // Задание 3  private Task StatAnalys()  {  const int Bsize = 512; // Размер битмапа  const int Mgn = 40; // Сдвиг от края битмапа  const int PointRadius = 2; // Размер точки на битмапе  const int TimeWidth = 50; // Ширина окна (количество записей для интерполяции)  int CurrentNode = NodesComboBox.SelectedIndex; // Какой узел отобразить  int TimeCount = NODES[CurrentNode].TimeStamps.Length; // Сколько в узле всего записей по времени  sliding.Maximum = TimeCount - TimeWidth;  debug.Text = "Max sl is:" + sliding.Maximum.ToString() + " valur sl is:" + sliding.Value;  //SlidingWindowTrackBar.Maximum = TimeCount - TimeWidth; // До куда можно двигать синее скользящее окно  //int TimeStart = SlidingWindowTrackBar.Value; // Время начала интерполяции  int TimeStart = sliding.Value; // Время начала интерполяции  int TimeEnd = TimeStart + TimeWidth; // Время конца интерполяции  double Shift = 1 / (double)40; // Шаг между точками, которые нужно вставить для интерполяции  double PrevValue = 0; // Предыдущее значение по оси Y, нужно для интерполяции  Bitmap b = new Bitmap(Bsize, Bsize);  Graphics g = Graphics.FromImage(b);  g.SmoothingMode = SmoothingMode.HighQuality;  g.InterpolationMode = InterpolationMode.HighQualityBicubic;  g.TextRenderingHint = TextRenderingHint.AntiAlias;  // Фон  g.FillRectangle(Brushes.White, 0, 0, Bsize, Bsize);  // Координатные оси  g.DrawLine(new Pen(Color.Black), Mgn, Mgn, Mgn, Bsize - Mgn);  g.DrawLine(new Pen(Color.Black), Mgn, Bsize - Mgn, Bsize - Mgn, Bsize - Mgn);  g.DrawString("Time", new Font("Arial", 10, FontStyle.Bold), Brushes.Black, Bsize - Mgn \* 2, Bsize - Mgn);  g.RotateTransform(90);  g.DrawString("Flow between nodes", new Font("Arial", 10, FontStyle.Bold), Brushes.Black, Mgn, -Mgn);  g.RotateTransform(-90);  // Данные с потока между узлами  for (int time = 0; time < TimeCount; time++)  {  int value = NODES[CurrentNode].Values[time]; // Значение в потоке в момент времени time  // Временная отметка в зоне интерполяции (в синем скользящем окне)  if (time > TimeStart && time < TimeEnd)  {  // Сколько дополнительных точек внедрить между существующими  for (int i = 0; i < 40; i++)  {  double NewTime = time - 1 + i \* Shift;  double NewValue = Interpolate(NewTime, time - 1, time, PrevValue, value);  int TimeToX = (int)ConvertRange(0, TimeCount, Mgn, Bsize - Mgn, NewTime);  int TimeToY = (int)ConvertRange(0, 255, Bsize - Mgn, Mgn, NewValue);  g.FillRectangle(Brushes.Red, TimeToX, TimeToY - PointRadius, PointRadius, PointRadius);  }  }  PrevValue = value;  // Визуализация на битмапе  int x = (int)ConvertRange(0, TimeCount, Mgn, Bsize - Mgn, time);  int y = (int)ConvertRange(0, 255, Bsize - Mgn, Mgn, NODES[CurrentNode].Values[time]);  g.FillRectangle(Brushes.Red, x, y - PointRadius, PointRadius, PointRadius);  }  // Скользящее окно  int WinX = (int)ConvertRange(0, TimeCount, Mgn, Bsize - Mgn, TimeStart);  int WinWidth = (int)ConvertRange(0, TimeCount, 0, Bsize, TimeWidth);  g.FillRectangle(new SolidBrush(Color.FromArgb(100, Color.Gray)), WinX, 0, WinWidth - PointRadius, Bsize);  g.Dispose();  b.Save(TOP\_PATH);  return Task.CompletedTask;  }  // Задание 4  private Task ThreeParamsDependency()  {  if (NODES == null) return null;  GetZHeightsAndValues();  float MaxHeight = Z\_HEIGHTS.Flatten().GetMax();  opengl1.Begin(OpenGL.GL\_QUADS);  const float size = 1.0f;  const float y = -5.0f;  for (int z = -5; z < 5; z++)  for (int x = -5; x < 5; x++)  {  float Height = Z\_HEIGHTS[x + 5, z + 5];  float Hue = (float)ConvertRange(0, MaxHeight, 0.2, 1, Height);  opengl1.Color(Hue, 0.0f, Hue);  Height /= 20.0f;  if (Height == 0) Height = 0.2f;  // Левая грань  opengl1.Vertex(x \* size, y, z \* size);  opengl1.Vertex(x \* size, y + Height, z \* size);  opengl1.Vertex(x \* size, y + Height, z \* size + size);  opengl1.Vertex(x \* size, y, z \* size + size);  // Задняя грань  opengl1.Vertex(x \* size, y, z \* size);  opengl1.Vertex(x \* size + size, y, z \* size);  opengl1.Vertex(x \* size + size, y, z \* size + size);  opengl1.Vertex(x \* size, y, z \* size + size);  // Нижняя грань  opengl1.Vertex(x \* size, y, z \* size);  opengl1.Vertex(x \* size, y + Height, z \* size);  opengl1.Vertex(x \* size + size, y + Height, z \* size);  opengl1.Vertex(x \* size + size, y, z \* size);  // Правая грань  opengl1.Vertex(x \* size + size, y, z \* size);  opengl1.Vertex(x \* size + size, y + Height, z \* size);  opengl1.Vertex(x \* size + size, y + Height, z \* size + size);  opengl1.Vertex(x \* size + size, y, z \* size + size);  // Верхняя грань  opengl1.Vertex(x \* size, y, z \* size + size);  opengl1.Vertex(x \* size, y + Height, z \* size + size);  opengl1.Vertex(x \* size + size, y + Height, z \* size + size);  opengl1.Vertex(x \* size + size, y, z \* size + size);  // Передняя грань  opengl1.Vertex(x \* size, y + Height, z \* size + size);  opengl1.Vertex(x \* size + size, y + Height, z \* size + size);  opengl1.Vertex(x \* size + size, y + Height, z \* size);  opengl1.Vertex(x \* size, y + Height, z \* size);  }  opengl1.End();  return Task.CompletedTask;  }  // Для задания 4 и 5  private Task GetZHeightsAndValues()  {  byte[] OldXs = NODES[XComboBox.SelectedIndex].Values;  byte[] OldYs = NODES[YComboBox.SelectedIndex].Values;  byte[] OldZs = NODES[ZComboBox.SelectedIndex].Values;  byte[] Xs = new byte[0];  byte[] Ys = new byte[0];  byte[] Zs = new byte[0];  // Фильтруем данные по времени  int TimeFromIndex = TimeFromComboBox.SelectedIndex;  int TimeToIndex = TimeToComboBox.SelectedIndex;  if (TimeToIndex < TimeFromIndex)  {  TimeFromIndex = 0;  TimeFromComboBox.SelectedIndex = 0;  TimeToIndex = OldXs.Length - 1;  TimeToComboBox.SelectedIndex = OldXs.Length - 1;  }  for (int i = TimeFromIndex; i < TimeToIndex; i++)  {  Xs = Xs.Add(OldXs[i]);  Ys = Ys.Add(OldYs[i]);  Zs = Zs.Add(OldZs[i]);  }  byte MinX = Xs.GetMin();  byte MaxX = Xs.GetMax();  byte MinY = Ys.GetMin();  byte MaxY = Ys.GetMax();  Z\_HEIGHTS = new int[10, 10];  Z\_VALUES = new int[10, 10];  int[,] Duplicates = new int[10, 10];  for (int y = 0; y < 10; y++)  for (int x = 0; x < 10; x++)  Duplicates[x, y] = 1;  for (int y = 0; y < 10; y++)  for (int x = 0; x < 10; x++)  Z\_VALUES[x, y] = -1;  for (int i = 0; i < Xs.Length; i++)  {  int x = 0;  int y = 0;  if (!(MinX == MaxX || MinY == MaxY))  {  x = (int)ConvertRange(MinX, MaxX, 0, 10, Xs[i]);  y = (int)ConvertRange(MinY, MaxY, 0, 10, Ys[i]);  } else  {  if (Ys[i] == MaxY)  y = 10;  if (Xs[i] == MaxX)  x = 10;  }  if (x == 10) x -= 1;  if (y == 10) y -= 1;  Duplicates[x, y]++;  Z\_HEIGHTS[x, y] += Zs[i];  Z\_VALUES[x, y] = Zs[i];  }  int PrevValue = 0;  for (int y = 0; y < 10; y++)  for (int x = 0; x < 10; x++)  if (Z\_VALUES[x, y] != -1)  PrevValue = Z\_VALUES[x, y];  for (int y = 0; y < 10; y++)  for (int x = 0; x < 10; x++)  {  if (Z\_VALUES[x, y] == -1)  Z\_VALUES[x, y] = PrevValue;  PrevValue = Z\_VALUES[x, y];  }  // Усредняем Z  for (int y = 0; y < 10; y++)  for (int x = 0; x < 10; x++)  Z\_HEIGHTS[x, y] /= Duplicates[x, y];  return Task.CompletedTask;  }  private static double Interpolate(double x, double x0, double x1, double y0, double y1)  {  if ((x1 - x0) == 0)  {  return (y0 + y1) / 2;  }  return y0 + (x - x0) \* (y1 - y0) / (x1 - x0);  }  private static double ConvertRange(double originalStart, double originalEnd, double newStart, double newEnd, double value)  {  double scale = (double)(newEnd - newStart) / (originalEnd - originalStart);  return newStart + ((value - originalStart) \* scale);  }  // Задание 5  private Task ScatterPlot()  {  GetZHeightsAndValues();  const int Bsize = 512; // Размер битмапа  const int Mgn = 41; // Сдвиг от края битмапа  const int IntervalSize = (Bsize - Mgn \* 2) / 10; // Размер интервала на битмапе  const int BarPointRadius = 3; // Размер точки деления  Bitmap b = new Bitmap(Bsize, Bsize);  Graphics g = Graphics.FromImage(b);  g.SmoothingMode = SmoothingMode.HighQuality;  g.InterpolationMode = InterpolationMode.HighQualityBicubic;  g.TextRenderingHint = TextRenderingHint.AntiAlias;  // Фон  g.FillRectangle(Brushes.White, 0, 0, Bsize, Bsize);  // Координатные оси  g.DrawLine(new Pen(Color.Black), Mgn, Mgn, Mgn, Bsize - Mgn); // Y  g.DrawLine(new Pen(Color.Black), Mgn, Bsize - Mgn, Bsize - Mgn, Bsize - Mgn); // X  Font font = new Font("Arial", 10, FontStyle.Bold);  g.DrawString("X", font, Brushes.Black, Bsize - Mgn / 2, Bsize - Mgn);  g.DrawString("Y", font, Brushes.Black, Mgn / 2, Mgn / 2);  // Деления  for (int i = 0; i < 10; i++)  {  // Ось Х  int x = Mgn + IntervalSize \* i + IntervalSize - BarPointRadius;  int y = Bsize - Mgn - BarPointRadius;  string text = (i + 1).ToString();  g.FillEllipse(Brushes.Black, x, y, BarPointRadius \* 2, BarPointRadius \* 2);  g.DrawString(text, font, Brushes.Black, x, Bsize - Mgn);  // Ось Y  x = Mgn - BarPointRadius;  y = Mgn + IntervalSize \* i - BarPointRadius;  text = (10 - i).ToString();  g.FillEllipse(Brushes.Black, x, y, BarPointRadius \* 2, BarPointRadius \* 2);  g.DrawString(text, font, Brushes.Black, Mgn / 2, Mgn + IntervalSize \* i);  }  // Начало координат  g.FillEllipse(Brushes.Black, Mgn - BarPointRadius, Bsize - Mgn - BarPointRadius, BarPointRadius \* 2, BarPointRadius \* 2);  g.DrawString("0", font, Brushes.Black, Mgn, Bsize - Mgn);  // Визуализация на битмапе  for (int y = 0; y < 10; y++)  for (int x = 0; x < 10; x++)  g.FillRectangle(new SolidBrush(Color.FromArgb(Z\_VALUES[x, y], Color.Black)), Mgn + IntervalSize \* x, Bsize - Mgn - IntervalSize - (IntervalSize \* y), IntervalSize, IntervalSize);  g.Dispose();  b.Save(BOTTOM\_PATH);  return Task.CompletedTask;  }  private Task DrawHistograms()  {  const int Bsize = 512; // Размер битмапа  Bitmap b\_xy = new Bitmap(Bsize, Bsize);  Bitmap b\_yz = new Bitmap(Bsize, Bsize);  Graphics g\_xy = Graphics.FromImage(b\_xy);  Graphics g\_yz = Graphics.FromImage(b\_yz);  g\_xy.SmoothingMode = SmoothingMode.HighQuality;  g\_yz.SmoothingMode = SmoothingMode.HighQuality;  // Фон  g\_xy.FillRectangle(Brushes.White, 0, 0, Bsize, Bsize);  g\_yz.FillRectangle(Brushes.White, 0, 0, Bsize, Bsize);  DensityCellXY = new int[10, 10];  DensityCellYZ = new int[10, 10];  int StepX = Bsize / 10;  int StepY = Bsize / 10;  int StepZ = Bsize / 10;  for (int i = 0; i < NODES[0].Values.Length; i++)  {  byte x = NODES[XComboBox.SelectedIndex].Values[i];  byte y = NODES[YComboBox.SelectedIndex].Values[i];  byte z = NODES[ZComboBox.SelectedIndex].Values[i];  double NewX = ConvertScale(0, 140, 0, Bsize, x);  double NewY = ConvertScale(0, 140, 0, Bsize, y);  double NewZ = ConvertScale(0, 140, 0, Bsize, z);  int ColumnX = (int)(NewX / StepX);  int ColumnZ = (int)(NewZ / StepZ);  int Row = 10 - (int)(NewY / StepY) - 1;  if (Row < 0) Row = 0;  if (Row > 9) Row = 9;  if (ColumnX < 0) ColumnX = 0;  if (ColumnX > 9) ColumnX = 9;  if (ColumnZ < 0) ColumnZ = 0;  if (ColumnZ > 9) ColumnZ = 9;  DensityCellXY[ColumnX, Row]++;  DensityCellYZ[ColumnZ, Row]++;  }  CalculateHistogramsXY(out int[] Xs, out int[] Ys);  int MaxX = Xs.GetMax();  int MaxY = Ys.GetMax();  for (int i = 0; i < 10; i++)  {  int Value = (int)ConvertScale(0, MaxY, 0, Bsize - StepY, Ys[i]);  g\_xy.FillRectangle(Brushes.Black, Bsize/2 + (StepX/2) \* i, 0, StepX/2, Value);  }  for (int i = 0; i < 10; i++)  {  int Value = (int)ConvertScale(0, MaxX, 0, (Bsize - StepX)/2, Xs[i]);  g\_xy.FillRectangle(Brushes.Black, 0, i \* StepY, Value, StepY);  }  g\_xy.FillRectangle(Brushes.Green, Bsize/2 - 5, 0, 10, Bsize);  CalculateHistogramsYZ(out int[] Zs, out int[] YZs);  int MaxZ = Zs.GetMax();  int MaxYZ = YZs.GetMax();  for (int i = 0; i < 10; i++)  {  int Value = (int)ConvertScale(0, MaxY, 0, Bsize - StepY, YZs[i]);  g\_yz.FillRectangle(Brushes.Black, Bsize / 2 + (StepZ / 2) \* i, 0, StepZ / 2, Value);  }  for (int i = 0; i < 10; i++)  {  int Value = (int)ConvertScale(0, MaxZ, 0, (Bsize - StepZ) / 2, Zs[i]);  g\_yz.FillRectangle(Brushes.Black, 0, i \* StepY, Value, StepY);  }  g\_yz.FillRectangle(Brushes.Green, Bsize / 2 - 5, 0, 10, Bsize);  g\_xy.Dispose();  g\_yz.Dispose();  b\_xy.Save(BACK\_PATH);  b\_yz.Save(LEFT\_PATH);  return Task.CompletedTask;  }  private void CalculateHistogramsXY(out int[] xs, out int[] ys)  {  xs = new int[10];  ys = new int[10];  for (int y = 0; y < DensityCellXY.GetLength(1); y++)  {  int SumX = 0;  int SumY = 0;  for (int x = 0; x < DensityCellXY.GetLength(0); x++)  {  SumX += DensityCellXY[x, y];  SumY += DensityCellXY[y, x];  }  xs[y] = SumX;  ys[y] = SumY;  }  }  private void CalculateHistogramsYZ(out int[] zs, out int[] ys)  {  zs = new int[10];  ys = new int[10];  for (int y = 0; y < DensityCellYZ.GetLength(1); y++)  {  int SumZ = 0;  int SumY = 0;  for (int z = 0; z < DensityCellYZ.GetLength(0); z++)  {  SumZ += DensityCellYZ[z, y];  SumY += DensityCellYZ[y, z];  }  zs[y] = SumZ;  ys[y] = SumY;  }  }  int[,] DensityCellXY;  int[,] DensityCellYZ;  public static double ConvertScale(double originalStart, double originalEnd, double newStart, double newEnd, double value)  {  return newStart + ((value - originalStart) \* ((double)(newEnd - newStart) / (originalEnd - originalStart)));  }  private Task PointsSync()  {  const int Bsize = 512; // Размер битмапа  const int PointRadius = 4; // Размер точки деления  Bitmap b\_xy = new Bitmap(Bsize, Bsize);  Bitmap b\_yz = new Bitmap(Bsize, Bsize);  Graphics g\_xy = Graphics.FromImage(b\_xy);  Graphics g\_yz = Graphics.FromImage(b\_yz);  g\_xy.SmoothingMode = SmoothingMode.HighQuality;  g\_yz.SmoothingMode = SmoothingMode.HighQuality;  // Фон  g\_xy.FillRectangle(Brushes.White, 0, 0, Bsize, Bsize);  g\_yz.FillRectangle(Brushes.White, 0, 0, Bsize, Bsize);  g\_xy.InterpolationMode = InterpolationMode.HighQualityBicubic;  g\_yz.InterpolationMode = InterpolationMode.HighQualityBicubic;  byte MinX = NODES[XComboBox.SelectedIndex].Values.GetMin();  byte MaxX = NODES[XComboBox.SelectedIndex].Values.GetMax();  byte MinY = NODES[YComboBox.SelectedIndex].Values.GetMin();  byte MaxY = NODES[YComboBox.SelectedIndex].Values.GetMax();  byte MinZ = NODES[ZComboBox.SelectedIndex].Values.GetMin();  byte MaxZ = NODES[ZComboBox.SelectedIndex].Values.GetMax();  for (int i = 0; i < NODES[0].Values.Length; i++)  {  byte x = NODES[XComboBox.SelectedIndex].Values[i];  byte y = NODES[YComboBox.SelectedIndex].Values[i];  byte z = NODES[ZComboBox.SelectedIndex].Values[i];  int gx = (int)ConvertRange(MinX, MaxX, 0, Bsize, x);  int gy = (int)ConvertRange(MinY, MaxY, 0, Bsize, z);  int gz = (int)ConvertRange(MinZ, MaxZ, 0, Bsize, y);  g\_xy.FillRectangle(Brushes.SteelBlue, gx - PointRadius, gy - PointRadius, PointRadius \* 2, PointRadius \* 2);  g\_yz.FillRectangle(Brushes.SteelBlue, gy - PointRadius, gz - PointRadius, PointRadius \* 2, PointRadius \* 2);  }  g\_xy.Dispose();  g\_yz.Dispose();  b\_xy.Save(FRONT\_PATH);  b\_yz.Save(RIGHT\_PATH);  return Task.CompletedTask;  }  // Задание 2  private void LoadCANFileButton\_Click(object sender, EventArgs e)  {  try  {  DialogResult res = openFileDialog1.ShowDialog();  if (res == DialogResult.OK)  {  byte[] Data = File.ReadAllBytes(openFileDialog1.FileName);  PACKAGES = new Package[0];  NODES = new Node[0];  // Определение уникальных узлов  HashSet<string> UniqueNodes = new HashSet<string>();  // Заполняем пакеты  for (int i = 0; i < Data.Length; i += 19)  {  byte[] TimeBytes = { Data[i], Data[i + 1] };  int Seconds = BitConverter.ToInt16(TimeBytes, 0);  string Time = TimeSpan.FromSeconds(Seconds).ToString();  byte Target = Data[i + 6];  byte Source = Data[i + 7];  byte Value = Data[i + 9];  UniqueNodes.Add(Source.ToString() + " -> " + Target.ToString());  PACKAGES = PACKAGES.Add(new Package(Time, Source, Target, Value));  }  // Заполняем узлы  foreach (var n in UniqueNodes)  {  Node node = new Node(n);  foreach (var p in PACKAGES)  if (node.BelongsToNode(p))  node.AddValue(p.Time, p.Value);  NODES = NODES.Add(node);  }  NodesComboBox.Items.AddRange(UniqueNodes.ToArray());  NodesComboBox.SelectedIndex = 0;  XComboBox.Items.AddRange(UniqueNodes.ToArray());  XComboBox.SelectedIndex = 0;  YComboBox.Items.AddRange(UniqueNodes.ToArray());  YComboBox.SelectedIndex = 1;  ZComboBox.Items.AddRange(UniqueNodes.ToArray());  ZComboBox.SelectedIndex = 2;  TimeFromComboBox.Items.AddRange(NODES[0].TimeStamps);  TimeFromComboBox.SelectedIndex = 0;  TimeToComboBox.Items.AddRange(NODES[0].TimeStamps);  TimeToComboBox.SelectedIndex = NODES[0].TimeStamps.Length - 1;  FillDataGrid(UniqueNodes);  StatAnalys();  DrawHistograms();  ScatterPlot();  DrawEverything();  }  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);  }  }  private void GetScreenshotButton\_Click(object sender, EventArgs e)  {  if (opengl1 == null) return;  DialogResult res = saveFileDialog1.ShowDialog();  try  {  if (res == DialogResult.OK)  {  Mat Screenshot = GetScreenshot(DataControl);  Screenshot.SaveImage(saveFileDialog1.FileName);  }  }  catch  {  MessageBox.Show("Something wrong with your picture");  }  }  private void DataControl\_MouseDown(object sender, MouseEventArgs e)  {  is\_rotate\_change = true;  start\_mouse\_pose = new int[] { e.X, e.Y };  }  private async void DataControl\_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();  await Task.Delay(1);  DrawEverything();  }  }  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 DataControl\_MouseUp(object sender, MouseEventArgs e)  {  is\_rotate\_change = false;  }  private async void DataControl\_MouseWheel(object sender, MouseEventArgs e)  {  if (e.Delta < 0)  distance\_z++;  else  distance\_z--;  await Task.Delay(1);  DrawEverything();  }  private void CheckBoxes\_CheckChanged(object sender, EventArgs e) => DrawEverything();  private async void ComboBoxes\_SelectedIndexChanged(object sender, EventArgs e)  {  if (PACKAGES == null || NODES == null) return;  if (NodesComboBox.SelectedIndex == -1) return;  if (XComboBox.SelectedIndex == -1) return;  if (YComboBox.SelectedIndex == -1) return;  if (ZComboBox.SelectedIndex == -1) return;  if (TimeFromComboBox.SelectedIndex == -1) return;  if (TimeToComboBox.SelectedIndex == -1) return;  if (TimeFromComboBox.SelectedIndex == TimeToComboBox.SelectedIndex) return;  StatAnalys();  await DrawHistograms();  await ScatterPlot();  //AveragePlot();  //MedianPlot();  await PointsSync();  DrawEverything();  }  private void angleY\_bar\_ValueChanged(object sender, EventArgs e)  {  if (!is\_rotate\_change)  {  angleY = (float)(((float)Math.PI / 180f) \* angleY\_bar.Value) + Math.PI;  DrawEverything();  }  }  private void angleX\_bar\_ValueChanged(object sender, EventArgs e)  {  if (!is\_rotate\_change)  {  angleX = (float)(((float)Math.PI / 180f) \* angleX\_bar.Value);  DrawEverything();  }  }  private void sliding\_ValueChanged(object sender, EventArgs e)  {  StatAnalys();  DrawEverything();  }  private void HideLeftPlaneCheckBox\_Click(object sender, EventArgs e)  {  if (HideTopPlaneCheckBox.Checked) StatAnalys();  DrawEverything();  }  }  public struct Package  {  public string Time { get; }  public byte Source { get; }  public byte Target { get; }  public byte Value { get; }  public Package(string time, byte source, byte target, byte value)  {  Time = time;  Source = source;  Target = target;  Value = value;  }  }  public class Node  {  public string Name { get; private set; }  public string[] TimeStamps { get; private set; }  public byte[] Values { get; private set; }  public Node(string name)  {  Name = name;  TimeStamps = new string[0];  Values = new byte[0];  }  public void AddValue(string time, byte value)  {  TimeStamps = TimeStamps.Add(time);  Values = Values.Add(value);  }  public bool BelongsToNode(Package package) =>  package.Source.ToString() + " -> " + package.Target.ToString() == Name;  }  } |