МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ автономное ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ



Кафедра СМАРТ-технологии

Лабораторная работа № 4:

«Многомерный анализ и визуализация мониторинговых данных»

По дисциплине: «Технологии визуализации данных систем управления»

Группа 221-327

№ группы

Студент Мезенцев И.В.

Подпись студента

Дата 27.05.2025

Дата сдачи

Преподаватель Идиатуллов Т.Т

Подпись преподавателя

2025

**Цель работы:**

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

**Задачи:**

1. Реализовать визуализацию сцены для отображения пространственных данных в виде куба с окружающими поверхностями для отображения графиков.

2. Реализовать управление отображением с помощью мыши с центром вращения сцены в геометрическом центре куба. Реализовать возможность скрывать куб или отдельные поверхности для графиков.

3. 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.

4. Выполнить статистический анализ записей как набора точек в многомерном

пространстве, вычисляя параметры описательной статистики методом скользящих сводных параметров.

5. Реализовать возможность задать глубину истории рассчитываемых параметров.

6. Разработать систему накопления данных для отображения зависимостей трех параметров по выбору начиная с временной отсечки и в течение заданного времени накопления. Третий параметр учитывается как среднее значение на заданном интервале двух других параметров за указанный период накопления.

7. Реализовать систему накопления данных для анализа зависимости трех параметров за заданный период времени, методом расчета описательной статистики (скейтерплоты) с фиксированной глубиной истории и без анализа выбросов.

8. Выполнить генерацию битовых карт с графиками описательной статистики(гистограммы и скейтерплотами) для заданных параметров и реализовать их использование в качестве текстур для размещения графиков на сцене визуа-лизации.

9. Реализовать систему отображения зависимости параметров как плоскости в кубе визуализации.

10. Реализовать возможность отображения графиков зависимостей между двумя параметрами (описательной статистики) на поверхностях для графиков, синхронизированно с осями куба визуализации

11. Реализовать систему визуализации данных (плоскости и графиков) в режиме скользящих сводных параметров.

12. Реализовать отображение системы координат и ребер описывающего куба с указанием визуализируемых параметров.

13. Реализовать сохранение полученного изображения в файл.

**Ход разработки проекта**

1. Инициализация проекта и настройка интерфейса

Проект был создан как приложение Windows Forms на языке C# с использованием библиотеки SharpGL для реализации 3D-графики. Основной элемент управления для отображения 3D-сцены – `sharpGLControl1`. Интерфейс пользователя включает элементы для загрузки данных (`btnLoadData`), выбора параметров для осей X, Y, Z основного 3D-графика (`cmbXAxis`, `cmbYAxis`, `cmbZAxis`), выбора типов графиков для каждой пары граней куба (`cmbFaceTypeX`, `cmbFaceTypeY`, `cmbFaceTypeZ`), а также выбора параметров для специфических типов графиков на гранях (например, `cmbTimePlotParamFace`, `cmbOutlierParamFace`). Также добавлены элементы управления для отображения табличных данных (`dataGridView1`, `dataGridViewRaw`) и сохранения изображения (`btnSaveImage`).

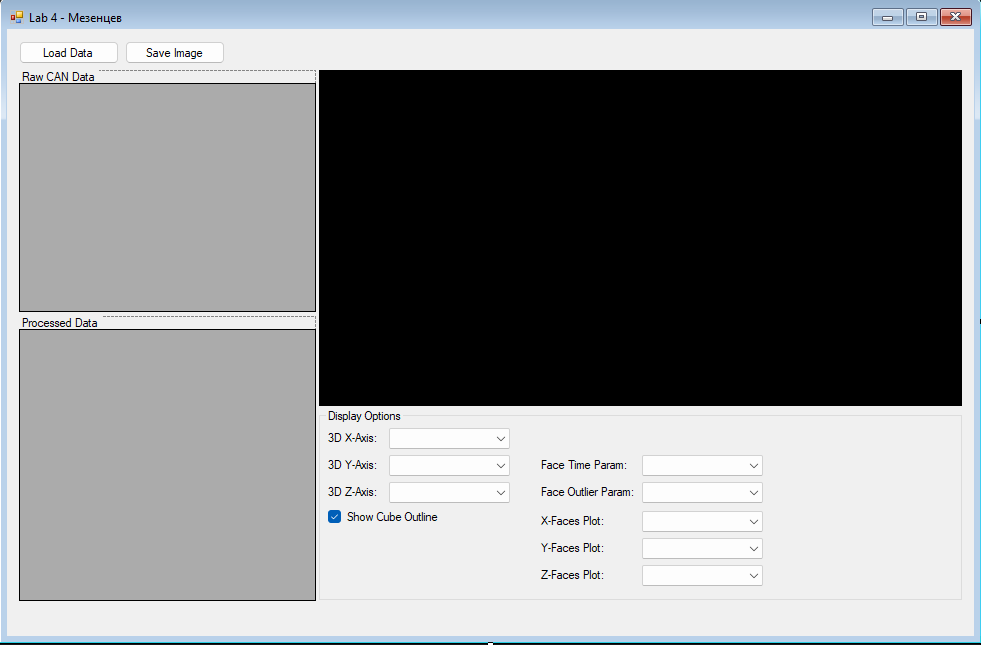


Рисунок 1 – Общий вид интерфейса приложения

2. Загрузка и обработка данных

Загрузка данных осуществляется по нажатию кнопки "Load Data". Поддерживается загрузка бинарных файлов, представляющих собой дамп сообщений CAN-шины. Метод `btnLoadData\_Click` открывает диалог выбора файла. После выбора файла, его содержимое считывается и парсится. Каждое CAN-сообщение (фрейм) имеет фиксированную структуру (17 байт в данном случае, с разделителями FF FF).

Метод `dataTableFromCAN` преобразует список загруженных `CANDumpData` объектов в `DataTable` (`CANdataTable`), где каждая строка соответствует одному CAN-фрейму, а столбцы – полям этого фрейма (TickStamp, Source, Dest, DLC, байты данных b1-b8 и т.д.).

Затем метод `FillTAble` выполняет агрегацию и преобразование этих данных. Он создает новую таблицу `dataTable`, где строки соответствуют уникальным временным меткам (`TickStamp`), а столбцы – параметрам, идентифицируемым как "Источник=>Приемник" (например, "10=>20"). Значением параметра берется первый байт данных (`b1`) соответствующего CAN-сообщения. Эта таблица используется для построения всех графиков. Обе таблицы (`CANdataTable` и `dataTable`) отображаются в элементах `DataGridView` для контроля (Рисунок 2).

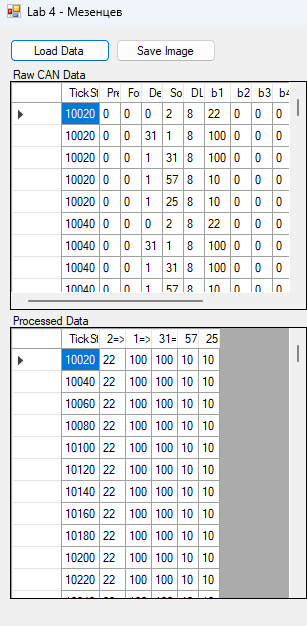


Рисунок 2 – Интерфейс с загруженными и обработанными данными в таблицах

После загрузки и обработки данных обновляются выпадающие списки (`ComboBox`) для выбора параметров визуализации с помощью метода `UpdateParameterComboBoxes`.

3. Настройка 3D-сцены и камеры

Инициализация OpenGL происходит в обработчике `sharpGLControl1\_OpenGLInitialized`, где устанавливается цвет очистки экрана и включается тест глубины. В `sharpGLControl1\_Resize` настраивается область вывода и матрица проекции (перспективная).

Основная отрисовка сцены выполняется в `sharpGLControl1\_OpenGLDraw`. Сначала очищается буфер цвета и глубины. Затем устанавливается камера с помощью `gl.Translate` (удаление камеры) и `gl.Rotate` (вращение на основе переменных `rotationX`, `rotationY`, изменяемых мышью). Реализованы стандартные элементы управления камерой: вращение сцены зажатой левой кнопкой мыши (`sharpGLControl1\_MouseDown`, `sharpGLControl1\_MouseMove`, `sharpGLControl1\_MouseUp`) и масштабирование (изменение `cameraDistance`) колесом мыши (`sharpGLControl1\_MouseWheel`).

4. Отрисовка базовых элементов сцены и основного 3D-графика

В методе `sharpGLControl1\_OpenGLDraw` последовательно отрисовываются:

Оси координат X (красная), Y (зеленая), Z (синяя) с помощью метода `DrawAxes`.

Ограничивающий проволочный куб (если включен чекбокс `chkShowCube`) с помощью `DrawBoundingWireCube`. Размер куба определяется константой `CUBE\_HALF\_SIZE`.

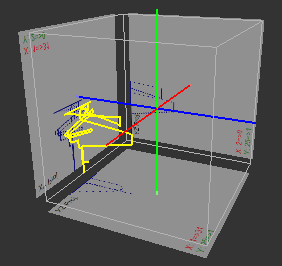


Рисунок 3 – 3D-сцена с осями и ограничивающим кубом

Для отрисовки основного 3D-графика данные подготавливаются в методе `PrepareDataAndTextures`. На основе выбранных в `cmbXAxis`, `cmbYAxis`, `cmbZAxis` параметров из `dataTable` формируется список точек `active3DDataPoints` (тип `List`). Значения параметров нормализуются методом `NormalizeValue` в диапазон `[-CUBE\_HALF\_SIZE, CUBE\_HALF\_SIZE]`. Метод `Draw3DLinePlot` отрисовывает эти точки как ломаную линию желтого цвета. Если в данных есть пропуски (невалидные точки), линия прерывается (Рисунок 4).

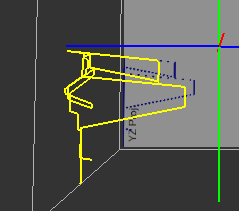


Рисунок 4 – Основной 3D-график временного ряда

5. Реализация проекционных плоскостей (граней куба)

Ключевой особенностью является отображение 2D-графиков на гранях виртуального куба. Эти грани отрисовываются методом `DrawCustomizableProjectionPlanes`. Каждая грань (точнее, пара противоположных граней) может отображать один из следующих типов графиков, выбираемых через `cmbFaceTypeX/Y/Z`:

`FacePlotType.None`: Грань не отображается.

`FacePlotType.Projection`: 2D-проекция основного 3D-графика.

`FacePlotType.TimePlot`: График зависимости выбранного параметра от времени.

`FacePlotType.OutlierPlot`: График распределения (точечная диаграмма) выбранного параметра.

Метод `DrawSingleFace` отвечает за отрисовку одной текстурированной грани. Он использует отсечение невидимых граней (back-face culling), проверяя Z-компоненту нормали грани в пространстве вида. Грани смещаются от центра вдоль их нормалей на величину `FACE\_OFFSET` для предотвращения Z-fighting и улучшения визуального восприятия (Рисунок 5).

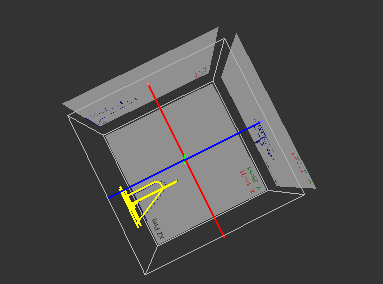


Рисунок 5 – Визуализация смещенных граней с проекциями

Текстуры для граней генерируются "на лету" при изменении данных или настроек. Все текстуры создаются с использованием GDI+ на объекте `Bitmap` размером `TEXTURE\_SIZE` x `TEXTURE\_SIZE`, а затем преобразуются в OpenGL текстуру методом `CreateTextureFromBitmap`.

Проекции 3D-графика (`CreateScatterPlotTexture`): Для граней X (PX, NX) создается проекция на плоскость YZ, для Y (PY, NY) – на XZ, для Z (PZ, NZ) – на XY. Точки из `active3DDataPoints` соответствующим образом проецируются и отрисовываются на `Bitmap`(Рисунок 6).

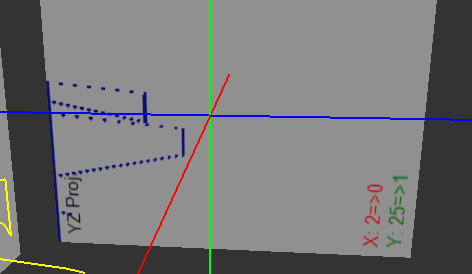


Рисунок 6 – Пример текстуры с проекцией YZ на грань X

Графики параметра от времени (`CreateTimePlotTexture`): Для выбранного параметра (из `cmbTimePlotParamFace` или аналогичных) и временных меток (`TickStamp`) из `dataTable` строится линейный график. Ось X представляет время (от `minTick` до `maxTick`), ось Y – значение параметра (от 0 до 255). На текстуре также рисуются оси, сетка и подписи (Рисунок 7).

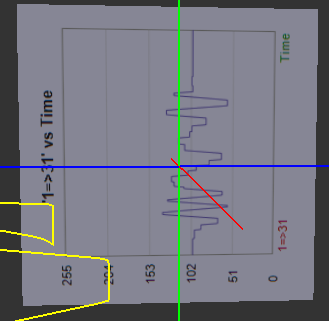


Рисунок 7 – Пример текстуры графика параметра от времени

Графики распределения/выбросов (`CreateOutlierPlotTexture`): Для выбранного параметра (из `cmbOutlierParamFace` или аналогичных) из `dataTable` строится одномерная точечная диаграмма (strip plot). Точки располагаются вдоль оси Y (значение параметра), а по оси X вносится небольшой случайный "джиттер" для лучшей визуализации плотности (Рисунок 8).

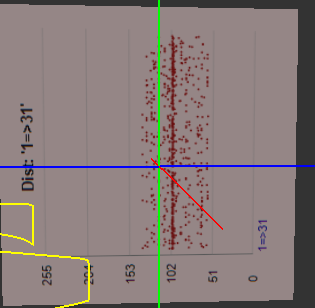


Рисунок 8 – Пример текстуры графика распределения параметра

Метод `PrepareDataAndTextures` координирует обновление `active3DDataPoints` и перегенерацию всех текстур для граней на основе текущих выбранных параметров и типов графиков. Привязка текстур к соответствующим граням происходит в `DrawCustomizableProjectionPlanes` и `DrawSingleFace`.

6. Реализация поверхностной 3D-гистограммы

Для анализа плотности распределения данных в проекции на плоскость XY была добавлена новая функция — поверхностная 3D-гистограмма. Её отображение включается с помощью специального чекбокса на форме.

Логика работы реализована в двух основных методах: PrepareHistogramDataXY и DrawHistogramSurface.

Вся гистограмма смещается вверх по оси Z и отрисовывается над основным кубом, предоставляя дополнительный слой аналитической информации о распределении данных (Рисунок 9).

7. Сохранение изображения

Функция сохранения текущего вида 3D-сцены реализована в методе `btnSaveImage\_Click`. Он использует `gl.ReadPixels` для считывания содержимого буфера кадра (`OpenGL.GL\_FRONT`) в `Bitmap`, который затем сохраняется в выбранный пользователем файл. Перед сохранением изображение переворачивается по вертикали, так как системы координат OpenGL и GDI+ имеют разное направление оси Y.

Вывод

В ходе выполнения курсовой работы было разработано приложение на языке C# с использованием библиотеки SharpGL, предназначенное для интерактивной трехмерной визуализации многопараметрических данных. Приложение успешно выполняет загрузку и парсинг бинарных данных (имитирующих CAN-дамп), обрабатывает их и преобразует в формат, пригодный для визуализации.

Реализовано отображение основного 3D-графика временного ряда по трем выбранным пользователем параметрам. Ключевой особенностью является возможность вывода различных 2D-графиков (проекций основного графика, графиков зависимости параметра от времени, графиков распределения параметра) на смещенные грани виртуального ограничивающего куба. Для этого был разработан механизм генерации текстур "на лету" с использованием GDI+ и их наложения на грани в 3D-сцене.

Пользовательский интерфейс позволяет гибко настраивать параметры визуализации и типы отображаемых графиков. Реализованы стандартные функции интерактивного управления камерой (вращение, масштабирование) и сохранения полученного изображения в файл. Разработанное программное обеспечение демонстрирует эффективный подход к комбинированию трехмерной визуализации с дополнительной 2D-аналитикой для исследования сложных многомерных данных.

**Приложение А**

**Листинг кода**

**using SharpGL;**

**using SharpGL.Enumerations;**

**using SharpGL.SceneGraph; // Для Vertex (если это SharpGL.Vertex)**

**// using SharpGL.Mathematics; // Удалено, так как вызывало ошибку в вашем случае**

**using System;**

**using System.Collections.Generic;**

**using System.Data;**

**using System.Drawing;**

**using System.Drawing.Imaging;**

**using System.IO;**

**using System.Linq;**

**using System.Windows.Forms;**

**// Marshal не используется активно, можно закомментировать, если не нужен для чего-то еще**

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

**namespace L4\_Four**

**{**

**public partial class Form1 : Form**

**{**

**// --- Data Members ---**

**DataTable CANdataTable;**

**DataTable dataTable; // Processed data for plotting (TickStamp, Param1, Param2, ...)**

**// --- OpenGL related members ---**

**float rotationX = 20.0f, rotationY = -30.0f;**

**float cameraDistance = 5.0f;**

**Point lastMousePos;**

**bool isMouseDown = false;**

**// Data for 3D plot**

**List<Point3F> active3DDataPoints = new List<Point3F>();**

**string selectedXParam, selectedYParam, selectedZParam;**

**// Textures for cube faces**

**Dictionary<string, uint> projectionTextures = new Dictionary<string, uint>();**

**// Keys: "PX", "NX", "PY", "NY", "PZ", "NZ"**

**// Parameters for face plots**

**string selectedTimePlotParamFace;**

**string selectedOutlierPlotParamFace;**

**// Constants**

**const int TEXTURE\_SIZE = 256;**

**const float CUBE\_HALF\_SIZE = 1.0f;**

**const float FACE\_OFFSET = 0.15f;**

**// Enum for face plot types**

**public enum FacePlotType**

**{**

**None,**

**Projection,**

**OutlierPlot,**

**TimePlot**

**}**

**// --- Data Structures ---**

**public struct CANDumpData**

**{**

**public UInt32 TickStamp;**

**public byte Prefix;**

**public byte Format;**

**public byte Dest;**

**public byte Source;**

**public byte DLC;**

**public byte b1, b2, b3, b4, b5, b6, b7, b8;**

**public CANDumpData(uint ts, byte p, byte f, byte d, byte s, byte dl, byte B1, byte B2, byte B3, byte B4, byte B5, byte B6, byte B7, byte B8)**

**{**

**TickStamp = ts; Prefix = p; Format = f; Dest = d; Source = s; DLC = dl;**

**b1 = B1; b2 = B2; b3 = B3; b4 = B4; b5 = B5; b6 = B6; b7 = B7; b8 = B8;**

**}**

**}**

**public struct Point3F**

**{**

**public float X, Y, Z;**

**public bool IsValid;**

**public Point3F(float x, float y, float z, bool isValid = true)**

**{**

**X = x; Y = y; Z = z;**

**IsValid = isValid;**

**}**

**}**

**public Form1()**

**{**

**InitializeComponent();**

**InitializeFaceTypeComboBoxes();**

**// Subscribe to events**

**this.btnLoadData.Click += new System.EventHandler(this.btnLoadData\_Click);**

**this.btnSaveImage.Click += new System.EventHandler(this.btnSaveImage\_Click);**

**this.cmbXAxis.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**this.cmbYAxis.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**this.cmbZAxis.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**this.cmbFaceTypeX.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**this.cmbFaceTypeY.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**this.cmbFaceTypeZ.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**// Ensure these ComboBoxes exist in your designer with these exact names**

**// or update the names here to match your designer.**

**if (this.cmbTimePlotParamFace != null)**

**this.cmbTimePlotParamFace.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**if (this.cmbOutlierParamFace != null)**

**this.cmbOutlierParamFace.SelectedIndexChanged += new System.EventHandler(this.cmbMainOrFaceParams\_Changed);**

**this.chkShowCube.CheckedChanged += new System.EventHandler(this.chkShowCube\_CheckedChanged);**

**}**

**private void InitializeFaceTypeComboBoxes()**

**{**

**var plotTypes = Enum.GetValues(typeof(FacePlotType)).Cast<FacePlotType>().ToList();**

**Action<ComboBox> setupCb = (cb) => {**

**if (cb != null) { cb.DataSource = plotTypes.ToList(); cb.SelectedItem = FacePlotType.Projection; }**

**};**

**setupCb(cmbFaceTypeX);**

**setupCb(cmbFaceTypeY);**

**setupCb(cmbFaceTypeZ);**

**}**

**// --- OpenGL Event Handlers ---**

**private void sharpGLControl1\_OpenGLInitialized(object sender, EventArgs e)**

**{**

**OpenGL gl = sharpGLControl1.OpenGL;**

**gl.ClearColor(0.2f, 0.2f, 0.2f, 1.0f);**

**gl.Enable(OpenGL.GL\_DEPTH\_TEST);**

**gl.Enable(OpenGL.GL\_TEXTURE\_2D);**

**}**

**private void sharpGLControl1\_Resize(object sender, EventArgs e)**

**{**

**OpenGL gl = sharpGLControl1.OpenGL;**

**gl.Viewport(0, 0, sharpGLControl1.Width, sharpGLControl1.Height);**

**SetProjection(gl);**

**}**

**private void SetProjection(OpenGL gl)**

**{**

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

**gl.LoadIdentity();**

**gl.Perspective(45.0f, (double)sharpGLControl1.Width / (double)sharpGLControl1.Height, 0.1f, 100.0f);**

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

**}**

**private void sharpGLControl1\_OpenGLDraw(object sender, RenderEventArgs args)**

**{**

**OpenGL gl = sharpGLControl1.OpenGL;**

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

**gl.LoadIdentity();**

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

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

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

**// 2. Отрисовка объектов сцены (в трансформированной системе координат)**

**DrawAxes(gl);**

**if (chkShowCube.Checked)**

**{**

**DrawBoundingWireCube(gl);**

**}**

**Draw3DLinePlot(gl); // <--- Ваш основной 3D график**

**DrawCustomizableProjectionPlanes(gl); // <--- Грани куба**

**gl.Flush();**

**}**

**// --- Data Processing and Normalization ---**

**private float NormalizeValue(byte val)**

**{**

**return ((float)val / 255.0f) \* (2.0f \* CUBE\_HALF\_SIZE) - CUBE\_HALF\_SIZE;**

**}**

**private void PrepareDataAndTextures()**

**{**

**OpenGL gl = sharpGLControl1.OpenGL;**

**active3DDataPoints.Clear();**

**foreach (var pair in projectionTextures)**

**{**

**if (pair.Value != 0) gl.DeleteTextures(1, new[] { pair.Value });**

**}**

**projectionTextures.Clear();**

**if (dataTable == null || cmbXAxis.SelectedItem == null || cmbYAxis.SelectedItem == null || cmbZAxis.SelectedItem == null)**

**{**

**sharpGLControl1.Invalidate();**

**return;**

**}**

**selectedXParam = cmbXAxis.SelectedItem.ToString();**

**selectedYParam = cmbYAxis.SelectedItem.ToString();**

**selectedZParam = cmbZAxis.SelectedItem.ToString();**

**if (!dataTable.Columns.Contains(selectedXParam) ||**

**!dataTable.Columns.Contains(selectedYParam) ||**

**!dataTable.Columns.Contains(selectedZParam))**

**{**

**sharpGLControl1.Invalidate();**

**return;**

**}**

**// Заполнение active3DDataPoints для основного 3D графика**

**foreach (DataRow row in dataTable.Rows)**

**{**

**byte rawX = 0, rawY = 0, rawZ = 0;**

**bool xValid = false, yValid = false, zValid = false;**

**if (row[selectedXParam] != DBNull.Value) { rawX = (byte)row[selectedXParam]; xValid = true; }**

**if (row[selectedYParam] != DBNull.Value) { rawY = (byte)row[selectedYParam]; yValid = true; }**

**if (row[selectedZParam] != DBNull.Value) { rawZ = (byte)row[selectedZParam]; zValid = true; }**

**System.Diagnostics.Debug.WriteLine($"Raw: X={rawX}, Y={rawY}, Z={rawZ}");**

**bool pointIsValid = xValid && yValid && zValid;**

**float normX = NormalizeValue(rawX);**

**float normY = NormalizeValue(rawY);**

**float normZ = NormalizeValue(rawZ);**

**// Debug output for normalized values:**

**System.Diagnostics.Debug.WriteLine($"Normalized: X={normX}, Y={normY}, Z={normZ}");**

**active3DDataPoints.Add(new Point3F(normX, normY, normZ, pointIsValid));**

**}**

**// Подготовка данных для текстур граней**

**if (!active3DDataPoints.Any()) // Если для 3D графика нет точек, то и для проекций не будет**

**{**

**sharpGLControl1.Invalidate();**

**return;**

**}**

**selectedTimePlotParamFace = cmbTimePlotParamFace?.SelectedItem?.ToString();**

**selectedOutlierPlotParamFace = cmbOutlierParamFace?.SelectedItem?.ToString();**

**List<PointF> xyPoints = active3DDataPoints.Where(p => p.IsValid).Select(p => new PointF(p.X, p.Y)).ToList();**

**List<PointF> xzPoints = active3DDataPoints.Where(p => p.IsValid).Select(p => new PointF(p.X, p.Z)).ToList();**

**List<PointF> yzPoints = active3DDataPoints.Where(p => p.IsValid).Select(p => new PointF(p.Y, p.Z)).ToList();**

**FacePlotType faceTypeX = cmbFaceTypeX?.SelectedItem != null ? (FacePlotType)cmbFaceTypeX.SelectedItem : FacePlotType.None;**

**ProcessFacePairTextures(gl, faceTypeX, "PX", "NX",**

**() => CreateScatterPlotTexture(gl, yzPoints, selectedYParam, selectedZParam, "YZ Proj."),**

**selectedOutlierPlotParamFace, selectedTimePlotParamFace);**

**FacePlotType faceTypeY = cmbFaceTypeY?.SelectedItem != null ? (FacePlotType)cmbFaceTypeY.SelectedItem : FacePlotType.None;**

**ProcessFacePairTextures(gl, faceTypeY, "PY", "NY",**

**() => CreateScatterPlotTexture(gl, xzPoints, selectedXParam, selectedZParam, "XZ Proj."),**

**selectedOutlierPlotParamFace, selectedTimePlotParamFace);**

**FacePlotType faceTypeZ = cmbFaceTypeZ?.SelectedItem != null ? (FacePlotType)cmbFaceTypeZ.SelectedItem : FacePlotType.None;**

**ProcessFacePairTextures(gl, faceTypeZ, "PZ", "NZ",**

**() => CreateScatterPlotTexture(gl, xyPoints, selectedXParam, selectedYParam, "XY Proj."),**

**selectedOutlierPlotParamFace, selectedTimePlotParamFace);**

**sharpGLControl1.Invalidate();**

**}**

**private void ProcessFacePairTextures(OpenGL gl, FacePlotType type, string keyPositive, string keyNegative,**

**Func<uint> projectionTextureFunc,**

**string outlierParam, string timeParam)**

**{**

**uint textureId = 0;**

**switch (type)**

**{**

**case FacePlotType.None: break;**

**case FacePlotType.Projection:**

**if (active3DDataPoints.Any() && !string.IsNullOrEmpty(selectedXParam) && !string.IsNullOrEmpty(selectedYParam) && !string.IsNullOrEmpty(selectedZParam))**

**textureId = projectionTextureFunc();**

**break;**

**case FacePlotType.OutlierPlot:**

**if (!string.IsNullOrEmpty(outlierParam) && dataTable != null && dataTable.Columns.Contains(outlierParam))**

**textureId = CreateOutlierPlotTexture(gl, outlierParam, $"Dist: '{outlierParam}'");**

**break;**

**case FacePlotType.TimePlot:**

**if (!string.IsNullOrEmpty(timeParam) && dataTable != null && dataTable.Columns.Contains(timeParam))**

**textureId = CreateTimePlotTexture(gl, timeParam, $"'{timeParam}' vs Time");**

**break;**

**}**

**if (textureId != 0) { projectionTextures[keyPositive] = textureId; projectionTextures[keyNegative] = textureId; }**

**}**

**// --- Texture Creation Methods ---**

**private uint CreateScatterPlotTexture(OpenGL gl, List<PointF> points, string xLabel, string yLabel, string title)**

**{**

**Bitmap bmp = new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE);**

**using (Graphics g = Graphics.FromImage(bmp))**

**{**

**g.Clear(Color.FromArgb(180, 240, 240, 240));**

**Font font = new Font("Arial", 10);**

**Brush dataBrush = Brushes.DarkBlue;**

**float textHeight = font.GetHeight(g);**

**g.DrawString(title, font, Brushes.Black, 5, 5);**

**if (!string.IsNullOrEmpty(xLabel)) g.DrawString($"X: {xLabel}", font, Brushes.Red, 5, TEXTURE\_SIZE - 2 \* textHeight - 5);**

**if (!string.IsNullOrEmpty(yLabel)) g.DrawString($"Y: {yLabel}", font, Brushes.Green, 5, TEXTURE\_SIZE - textHeight - 5);**

**float pointRadius = 1.5f;**

**if (points != null && points.Any())**

**{**

**foreach (var p in points)**

**{**

**float tx = (p.X + CUBE\_HALF\_SIZE) / (2 \* CUBE\_HALF\_SIZE) \* TEXTURE\_SIZE;**

**float ty = (p.Y + CUBE\_HALF\_SIZE) / (2 \* CUBE\_HALF\_SIZE) \* TEXTURE\_SIZE;**

**g.FillEllipse(dataBrush, tx - pointRadius, ty - pointRadius, 2 \* pointRadius, 2 \* pointRadius);**

**}**

**}**

**}**

**// Для отладки: bmp.Save($"debug\_scatter\_{title.Replace(" ", "")}.png");**

**return CreateTextureFromBitmap(gl, bmp);**

**}**

**private uint CreateTimePlotTexture(OpenGL gl, string paramName, string title)**

**{**

**if (dataTable == null || string.IsNullOrEmpty(paramName) || !dataTable.Columns.Contains(paramName) || !dataTable.Columns.Contains("TickStamp"))**

**return CreateTextureFromBitmap(gl, new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE)); // Возвращаем пустую текстуру**

**List<PointF> timeSeriesPoints = new List<PointF>();**

**UInt32 minTick = UInt32.MaxValue, maxTick = 0;**

**foreach (DataRow row in dataTable.Rows)**

**{**

**if (row[paramName] != DBNull.Value && row["TickStamp"] != DBNull.Value)**

**{**

**UInt32 tick = (UInt32)row["TickStamp"];**

**byte val = (byte)row[paramName];**

**timeSeriesPoints.Add(new PointF(tick, val));**

**if (tick < minTick) minTick = tick;**

**if (tick > maxTick) maxTick = tick;**

**}**

**}**

**if (timeSeriesPoints.Count < 2) return CreateTextureFromBitmap(gl, new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE));**

**Bitmap bmp = new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE);**

**using (Graphics g = Graphics.FromImage(bmp))**

**{**

**g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;**

**g.Clear(Color.FromArgb(180, 220, 220, 250)); // Semi-transparent background**

**Font axisFont = new Font("Arial", 8);**

**Font titleFont = new Font("Arial", 10, FontStyle.Bold);**

**Pen linePen = new Pen(Color.DarkSlateBlue, 1.5f);**

**Pen gridPen = new Pen(Color.FromArgb(100, 128, 128, 128), 0.5f); // Lighter grid**

**float padding = 15;**

**float headerHeight = g.MeasureString(title, titleFont).Height + 5;**

**float footerHeight = g.MeasureString("Time", axisFont).Height + 5;**

**float yAxisLabelWidth = g.MeasureString("255", axisFont).Width + 5;**

**RectangleF plotArea = new RectangleF(**

**padding + yAxisLabelWidth,**

**padding + headerHeight,**

**TEXTURE\_SIZE - (2 \* padding) - yAxisLabelWidth - 5, // Extra 5 for right margin**

**TEXTURE\_SIZE - (2 \* padding) - headerHeight - footerHeight - 5 // Extra 5 for bottom margin**

**);**

**if (plotArea.Width <= 0 || plotArea.Height <= 0) return CreateTextureFromBitmap(gl, bmp); // Too small**

**SizeF titleSize = g.MeasureString(title, titleFont);**

**g.DrawString(title, titleFont, Brushes.Black, plotArea.X + Math.Max(0, (plotArea.Width - titleSize.Width) / 2), padding);**

**g.DrawString(paramName, axisFont, Brushes.DarkRed, padding + yAxisLabelWidth, plotArea.Bottom + 2);**

**g.DrawString("Time", axisFont, Brushes.DarkGreen, plotArea.Right - g.MeasureString("Time", axisFont).Width, plotArea.Bottom + 2);**

**g.DrawRectangle(Pens.DimGray, plotArea.X, plotArea.Y, plotArea.Width, plotArea.Height);**

**for (int i = 0; i <= 5; i++) // Y-axis grid and labels**

**{**

**float yVal = i \* (255.0f / 5.0f);**

**float yPos = plotArea.Bottom - (yVal / 255.0f) \* plotArea.Height;**

**g.DrawLine(gridPen, plotArea.Left, yPos, plotArea.Right, yPos);**

**g.DrawString(Math.Round(yVal).ToString(), axisFont, Brushes.Black, padding, yPos - axisFont.Height / 2);**

**}**

**// Optional: X-axis grid (time)**

**// ...**

**List<PointF> screenPoints = new List<PointF>();**

**float tickRange = (maxTick - minTick);**

**if (tickRange == 0) tickRange = 1; // Avoid division by zero**

**foreach (var p in timeSeriesPoints.OrderBy(pt => pt.X)) // Sort by time**

**{**

**float tx = plotArea.X + ((p.X - minTick) / tickRange) \* plotArea.Width;**

**float ty = plotArea.Bottom - ((p.Y / 255.0f) \* plotArea.Height); // Y is inverted for GDI+**

**screenPoints.Add(new PointF(tx, ty));**

**}**

**if (screenPoints.Count > 1)**

**{**

**g.DrawLines(linePen, screenPoints.ToArray());**

**}**

**}**

**return CreateTextureFromBitmap(gl, bmp);**

**}**

**private uint CreateOutlierPlotTexture(OpenGL gl, string paramName, string title)**

**{**

**if (dataTable == null || string.IsNullOrEmpty(paramName) || !dataTable.Columns.Contains(paramName))**

**return CreateTextureFromBitmap(gl, new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE));**

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

**foreach (DataRow row in dataTable.Rows)**

**{**

**if (row[paramName] != DBNull.Value)**

**{**

**values.Add((byte)row[paramName]);**

**}**

**}**

**if (values.Count == 0) return CreateTextureFromBitmap(gl, new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE));**

**Bitmap bmp = new Bitmap(TEXTURE\_SIZE, TEXTURE\_SIZE);**

**using (Graphics g = Graphics.FromImage(bmp))**

**{**

**g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias;**

**g.Clear(Color.FromArgb(180, 250, 220, 220)); // Semi-transparent background**

**Font axisFont = new Font("Arial", 8);**

**Font titleFont = new Font("Arial", 10, FontStyle.Bold);**

**Brush dataBrush = Brushes.Maroon;**

**Pen gridPen = new Pen(Color.FromArgb(100, 128, 128, 128), 0.5f);**

**float padding = 15;**

**float headerHeight = g.MeasureString(title, titleFont).Height + 5;**

**float footerHeight = g.MeasureString(paramName, axisFont).Height + 5;**

**float yAxisLabelWidth = g.MeasureString("255", axisFont).Width + 5;**

**RectangleF plotArea = new RectangleF(**

**padding + yAxisLabelWidth,**

**padding + headerHeight,**

**TEXTURE\_SIZE - (2 \* padding) - yAxisLabelWidth - 5,**

**TEXTURE\_SIZE - (2 \* padding) - headerHeight - footerHeight - 5**

**);**

**if (plotArea.Width <= 0 || plotArea.Height <= 0) return CreateTextureFromBitmap(gl, bmp);**

**SizeF titleSize = g.MeasureString(title, titleFont);**

**g.DrawString(title, titleFont, Brushes.Black, plotArea.X + Math.Max(0, (plotArea.Width - titleSize.Width) / 2), padding);**

**g.DrawString(paramName, axisFont, Brushes.DarkBlue, plotArea.X, plotArea.Bottom + 2); // Y-axis title**

**g.DrawLine(Pens.DimGray, plotArea.X, plotArea.Y, plotArea.X, plotArea.Bottom); // Y-axis line**

**for (int i = 0; i <= 5; i++) // Y-axis grid and labels**

**{**

**float yVal = i \* (255.0f / 5.0f);**

**float yPos = plotArea.Bottom - (yVal / 255.0f) \* plotArea.Height;**

**g.DrawLine(gridPen, plotArea.Left, yPos, plotArea.Right, yPos); // Grid line**

**g.DrawString(Math.Round(yVal).ToString(), axisFont, Brushes.Black, padding, yPos - axisFont.Height / 2);**

**}**

**float pointSize = 2f;**

**Random rnd = new Random();**

**foreach (byte val in values)**

**{**

**float y = plotArea.Bottom - ((val / 255.0f) \* plotArea.Height);**

**float x = plotArea.X + 5 + (float)rnd.NextDouble() \* Math.Max(0, plotArea.Width - 10); // Jitter X**

**g.FillEllipse(dataBrush, x - pointSize / 2, y - pointSize / 2, pointSize, pointSize);**

**}**

**}**

**return CreateTextureFromBitmap(gl, bmp);**

**}**

**private uint CreateTextureFromBitmap(OpenGL gl, Bitmap bitmap)**

**{**

**bitmap.RotateFlip(RotateFlipType.RotateNoneFlipY);**

**uint[] textureIds = new uint[1];**

**gl.GenTextures(1, textureIds);**

**uint textureId = textureIds[0];**

**gl.BindTexture(OpenGL.GL\_TEXTURE\_2D, textureId);**

**gl.TexParameter(OpenGL.GL\_TEXTURE\_2D, OpenGL.GL\_TEXTURE\_WRAP\_S, OpenGL.GL\_CLAMP\_TO\_EDGE);**

**gl.TexParameter(OpenGL.GL\_TEXTURE\_2D, OpenGL.GL\_TEXTURE\_WRAP\_T, OpenGL.GL\_CLAMP\_TO\_EDGE);**

**gl.TexParameter(OpenGL.GL\_TEXTURE\_2D, OpenGL.GL\_TEXTURE\_MIN\_FILTER, OpenGL.GL\_LINEAR);**

**gl.TexParameter(OpenGL.GL\_TEXTURE\_2D, OpenGL.GL\_TEXTURE\_MAG\_FILTER, OpenGL.GL\_LINEAR);**

**BitmapData bmpData = bitmap.LockBits(new Rectangle(0, 0, bitmap.Width, bitmap.Height),**

**ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);**

**gl.TexImage2D(OpenGL.GL\_TEXTURE\_2D, 0, OpenGL.GL\_RGBA, bitmap.Width, bitmap.Height, 0,**

**OpenGL.GL\_BGRA, OpenGL.GL\_UNSIGNED\_BYTE, bmpData.Scan0);**

**bitmap.UnlockBits(bmpData);**

**bitmap.Dispose();**

**gl.BindTexture(OpenGL.GL\_TEXTURE\_2D, 0);**

**return textureId;**

**}**

**// --- Drawing Methods ---**

**private void DrawAxes(OpenGL gl)**

**{**

**float axisLength = CUBE\_HALF\_SIZE \* 1.2f;**

**gl.LineWidth(2.0f);**

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

**gl.Color(1.0f, 0.0f, 0.0f); gl.Vertex(-axisLength, 0.0f, 0.0f); gl.Vertex(axisLength, 0.0f, 0.0f);**

**gl.Color(0.0f, 1.0f, 0.0f); gl.Vertex(0.0f, -axisLength, 0.0f); gl.Vertex(0.0f, axisLength, 0.0f);**

**gl.Color(0.0f, 0.0f, 1.0f); gl.Vertex(0.0f, 0.0f, -axisLength); gl.Vertex(0.0f, 0.0f, axisLength);**

**gl.End();**

**gl.LineWidth(1.0f);**

**}**

**private void DrawBoundingWireCube(OpenGL gl)**

**{**

**gl.Color(0.7f, 0.7f, 0.7f, 0.5f);**

**gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_LINE); // ИСПРАВЛЕНО**

**float s = CUBE\_HALF\_SIZE;**

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

**gl.Vertex(s, s, s); gl.Vertex(-s, s, s); gl.Vertex(-s, -s, s); gl.Vertex(s, -s, s);**

**gl.Vertex(s, s, -s); gl.Vertex(s, -s, -s); gl.Vertex(-s, -s, -s); gl.Vertex(-s, s, -s);**

**gl.Vertex(s, s, s); gl.Vertex(s, s, -s); gl.Vertex(-s, s, -s); gl.Vertex(-s, s, s);**

**gl.Vertex(s, -s, s); gl.Vertex(-s, -s, s); gl.Vertex(-s, -s, -s); gl.Vertex(s, -s, -s);**

**gl.Vertex(s, s, s); gl.Vertex(s, -s, s); gl.Vertex(s, -s, -s); gl.Vertex(s, s, -s);**

**gl.Vertex(-s, s, s); gl.Vertex(-s, s, -s); gl.Vertex(-s, -s, -s); gl.Vertex(-s, -s, s);**

**gl.End();**

**gl.PolygonMode(OpenGL.GL\_FRONT\_AND\_BACK, OpenGL.GL\_FILL); // ИСПРАВЛЕНО: Reset polygon mode**

**}**

**private void Draw3DLinePlot(OpenGL gl)**

**{**

**if (active3DDataPoints.Count < 2) return;**

**gl.Color(1.0f, 1.0f, 0.0f); // Yellow**

**gl.LineWidth(1.5f);**

**gl.Begin(OpenGL.GL\_LINE\_STRIP);**

**foreach (var point in active3DDataPoints)**

**{**

**if (point.IsValid)**

**{**

**gl.Vertex(point.X, point.Y, point.Z);**

**}**

**else**

**{**

**gl.End(); gl.Begin(OpenGL.GL\_LINE\_STRIP);**

**}**

**}**

**gl.End();**

**gl.LineWidth(1.0f);**

**}**

**private void DrawCustomizableProjectionPlanes(OpenGL gl)**

**{**

**gl.Enable(OpenGL.GL\_BLEND);**

**gl.BlendFunc(OpenGL.GL\_SRC\_ALPHA, OpenGL.GL\_ONE\_MINUS\_SRC\_ALPHA);**

**float s = CUBE\_HALF\_SIZE; // 's' используется внутри лямбда-выражений drawQuadAction**

**double[] mv\_array = new double[16];**

**gl.GetDouble(OpenGL.GL\_MODELVIEW\_MATRIX, mv\_array);**

**// YZ Plot на PX грани (положительная X)**

**DrawSingleFace(gl, mv\_array, new Vertex(1, 0, 0), "PX", s, () => {**

**gl.TexCoord(0.0f, 1.0f); gl.Vertex(s, -s, -s);**

**gl.TexCoord(1.0f, 1.0f); gl.Vertex(s, s, -s);**

**gl.TexCoord(1.0f, 0.0f); gl.Vertex(s, s, s);**

**gl.TexCoord(0.0f, 0.0f); gl.Vertex(s, -s, s);**

**}, FACE\_OFFSET); // <--- Передаем FACE\_OFFSET**

**// YZ Plot на NX грани (отрицательная X)**

**DrawSingleFace(gl, mv\_array, new Vertex(-1, 0, 0), "NX", s, () => {**

**gl.TexCoord(0.0f, 1.0f); gl.Vertex(-s, -s, -s);**

**gl.TexCoord(1.0f, 1.0f); gl.Vertex(-s, s, -s);**

**gl.TexCoord(1.0f, 0.0f); gl.Vertex(-s, s, s);**

**gl.TexCoord(0.0f, 0.0f); gl.Vertex(-s, -s, s);**

**}, FACE\_OFFSET); // <--- Передаем FACE\_OFFSET**

**// XZ Proj. на PY грани (верхняя, положительная Y)**

**DrawSingleFace(gl, mv\_array, new Vertex(0, 1, 0), "PY", s, () => {**

**gl.TexCoord(0.0f, 1.0f); gl.Vertex(-s, s, -s);**

**gl.TexCoord(1.0f, 1.0f); gl.Vertex(s, s, -s);**

**gl.TexCoord(1.0f, 0.0f); gl.Vertex(s, s, s);**

**gl.TexCoord(0.0f, 0.0f); gl.Vertex(-s, s, s);**

**}, FACE\_OFFSET); // <--- Передаем FACE\_OFFSET**

**// XZ Proj. на NY грани (нижняя, отрицательная Y)**

**DrawSingleFace(gl, mv\_array, new Vertex(0, -1, 0), "NY", s, () => {**

**gl.TexCoord(1.0f, 1.0f); gl.Vertex(s, -s, -s);**

**gl.TexCoord(0.0f, 1.0f); gl.Vertex(-s, -s, -s);**

**gl.TexCoord(0.0f, 0.0f); gl.Vertex(-s, -s, s);**

**gl.TexCoord(1.0f, 0.0f); gl.Vertex(s, -s, s);**

**}, FACE\_OFFSET); // <--- Передаем FACE\_OFFSET**

**// XY Proj. на PZ грани (передняя, положительная Z)**

**DrawSingleFace(gl, mv\_array, new Vertex(0, 0, 1), "PZ", s, () => {**

**gl.TexCoord(0.0f, 1.0f); gl.Vertex(-s, -s, s);**

**gl.TexCoord(1.0f, 1.0f); gl.Vertex(s, -s, s);**

**gl.TexCoord(1.0f, 0.0f); gl.Vertex(s, s, s);**

**gl.TexCoord(0.0f, 0.0f); gl.Vertex(-s, s, s);**

**}, FACE\_OFFSET); // <--- Передаем FACE\_OFFSET**

**// XY Proj. на NZ грани (задняя, отрицательная Z)**

**DrawSingleFace(gl, mv\_array, new Vertex(0, 0, -1), "NZ", s, () => {**

**gl.TexCoord(0.0f, 1.0f); gl.Vertex(-s, -s, -s);**

**gl.TexCoord(1.0f, 1.0f); gl.Vertex(s, -s, -s);**

**gl.TexCoord(1.0f, 0.0f); gl.Vertex(s, s, -s);**

**gl.TexCoord(0.0f, 0.0f); gl.Vertex(-s, s, -s);**

**}, FACE\_OFFSET); // <--- Передаем FACE\_OFFSET**

**gl.BindTexture(OpenGL.GL\_TEXTURE\_2D, 0); // Сброс привязки текстуры**

**gl.Disable(OpenGL.GL\_BLEND);**

**}**

**private void DrawSingleFace(OpenGL gl, double[] mvMatrix, Vertex localNormal, string textureKey, float size, Action drawQuadAction, float offset)**

**{**

**if (!projectionTextures.TryGetValue(textureKey, out uint textureId) || textureId == 0) return;**

**float nx\_model = localNormal.X;**

**float ny\_model = localNormal.Y;**

**float nz\_model = localNormal.Z;**

**float viewNormalZ = (float)(mvMatrix[2] \* nx\_model + mvMatrix[6] \* ny\_model + mvMatrix[10] \* nz\_model);**

**if (viewNormalZ > 0.001f) return;**

**gl.Color(1.0f, 1.0f, 1.0f, 0.7f); // Например, 70% непрозрачности**

**gl.BindTexture(OpenGL.GL\_TEXTURE\_2D, textureId);**

**gl.PushMatrix(); // Сохраняем текущую матрицу ModelView**

**gl.Translate(localNormal.X \* offset, localNormal.Y \* offset, localNormal.Z \* offset);**

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

**drawQuadAction();**

**gl.End();**

**gl.PopMatrix();**

**}**

**// --- Mouse Controls for Camera ---**

**private void sharpGLControl1\_MouseDown(object sender, MouseEventArgs e)**

**{**

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

**{**

**isMouseDown = true;**

**lastMousePos = e.Location;**

**}**

**}**

**private void sharpGLControl1\_MouseMove(object sender, MouseEventArgs e)**

**{**

**if (isMouseDown)**

**{**

**float dx = e.X - lastMousePos.X;**

**float dy = e.Y - lastMousePos.Y;**

**rotationY += dx \* 0.5f;**

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

**lastMousePos = e.Location;**

**sharpGLControl1.Invalidate();**

**}**

**}**

**private void sharpGLControl1\_MouseUp(object sender, MouseEventArgs e)**

**{**

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

**{**

**isMouseDown = false;**

**}**

**}**

**private void sharpGLControl1\_MouseWheel(object sender, MouseEventArgs e)**

**{**

**cameraDistance -= e.Delta \* 0.005f;**

**if (cameraDistance < 1.0f) cameraDistance = 1.0f;**

**if (cameraDistance > 20.0f) cameraDistance = 20.0f;**

**sharpGLControl1.Invalidate();**

**}**

**// --- UI Event Handlers (Buttons, ComboBoxes, CheckBoxes) ---**

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

**{**

**if (openFileDialog1.ShowDialog() == DialogResult.Cancel)**

**return;**

**List<CANDumpData> frames = new List<CANDumpData>(); // Declare frames here**

**try**

**{**

**using (FileStream fs = new FileStream(openFileDialog1.FileName, FileMode.Open))**

**{**

**byte[] bytes = new byte[fs.Length];**

**fs.Read(bytes, 0, bytes.Length);**

**byte[] frameBuffer = new byte[17];**

**int counter = 0;**

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

**{**

**if (i + 1 < bytes.Length && bytes[i] == 255 && bytes[i + 1] == 255)**

**{**

**if (counter == 17)**

**{**

**UInt32 time = BitConverter.ToUInt32(frameBuffer, 0); // Simpler way**

**frames.Add(new CANDumpData(**

**time, frameBuffer[4], frameBuffer[5], frameBuffer[6], frameBuffer[7],**

**frameBuffer[8], frameBuffer[9], frameBuffer[10], frameBuffer[11],**

**frameBuffer[12], frameBuffer[13], frameBuffer[14], frameBuffer[15], frameBuffer[16]**

**));**

**}**

**counter = 0;**

**i++; // Skip the second FF**

**}**

**else**

**{**

**if (counter < 17)**

**{**

**frameBuffer[counter] = bytes[i];**

**counter++;**

**}**

**}**

**}**

**}**

**CANdataTable = dataTableFromCAN(frames); // Pass the populated frames list**

**dataGridView1.DataSource = CANdataTable; // Display raw CAN data**

**dataGridView1.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);**

**FillTAble(); // Process into the 'dataTable' for plotting**

**UpdateParameterComboBoxes();**

**PrepareDataAndTextures();**

**}**

**catch (Exception ex)**

**{**

**MessageBox.Show($"Error loading or processing data: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);**

**}**

**}**

**private void FillTAble()**

**{**

**dataTable = new DataTable();**

**dataTable.Columns.Add("TickStamp", typeof(UInt32));**

**if (CANdataTable == null || CANdataTable.Rows.Count == 0) return;**

**// Get unique, sorted TickStamps**

**var uniqueTickStamps = CANdataTable.AsEnumerable()**

**.Select(row => row.Field<UInt32>("TickStamp"))**

**.Distinct()**

**.OrderBy(t => t);**

**foreach (UInt32 tickStamp in uniqueTickStamps)**

**{**

**DataRow[] rowsForTick = CANdataTable.Select($"TickStamp = {tickStamp}");**

**DataRow newRow = dataTable.NewRow();**

**newRow["TickStamp"] = tickStamp;**

**foreach (DataRow canRow in rowsForTick)**

**{**

**string paramName = $"{canRow["Source"]}=>{canRow["Dest"]}";**

**if (!dataTable.Columns.Contains(paramName))**

**{**

**dataTable.Columns.Add(paramName, typeof(byte));**

**}**

**if (canRow["b1"] != DBNull.Value)**

**{**

**newRow[paramName] = (byte)canRow["b1"];**

**}**

**else**

**{**

**newRow[paramName] = DBNull.Value;**

**}**

**}**

**dataTable.Rows.Add(newRow);**

**}**

**dataGridViewRaw.DataSource = dataTable; // Display processed data**

**dataGridViewRaw.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCellsExceptHeader);**

**}**

**private DataTable dataTableFromCAN(List<CANDumpData> frames)**

**{**

**DataTable table = new DataTable();**

**table.Columns.Add("TickStamp", typeof(UInt32));**

**table.Columns.Add("Prefix", typeof(byte));**

**table.Columns.Add("Format", typeof(byte));**

**table.Columns.Add("Dest", typeof(byte));**

**table.Columns.Add("Source", typeof(byte));**

**table.Columns.Add("DLC", typeof(byte));**

**table.Columns.Add("b1", typeof(byte)); table.Columns.Add("b2", typeof(byte));**

**table.Columns.Add("b3", typeof(byte)); table.Columns.Add("b4", typeof(byte));**

**table.Columns.Add("b5", typeof(byte)); table.Columns.Add("b6", typeof(byte));**

**table.Columns.Add("b7", typeof(byte)); table.Columns.Add("b8", typeof(byte));**

**foreach (CANDumpData data in frames)**

**{**

**table.Rows.Add(data.TickStamp, data.Prefix, data.Format, data.Dest, data.Source, data.DLC,**

**data.b1, data.b2, data.b3, data.b4, data.b5, data.b6, data.b7, data.b8);**

**}**

**return table; // Ensure this return is always reached**

**}**

**private void UpdateParameterComboBoxes()**

**{**

**// Store current selections**

**string oldX = cmbXAxis.SelectedItem?.ToString();**

**string oldY = cmbYAxis.SelectedItem?.ToString();**

**string oldZ = cmbZAxis.SelectedItem?.ToString();**

**string oldTimeFace = cmbTimePlotParamFace?.SelectedItem?.ToString();**

**string oldOutlierFace = cmbOutlierParamFace?.SelectedItem?.ToString();**

**// Clear items**

**cmbXAxis.Items.Clear();**

**cmbYAxis.Items.Clear();**

**cmbZAxis.Items.Clear();**

**if (cmbTimePlotParamFace != null) cmbTimePlotParamFace.Items.Clear();**

**if (cmbOutlierParamFace != null) cmbOutlierParamFace.Items.Clear();**

**if (dataTable != null)**

**{**

**var paramNames = dataTable.Columns.Cast<DataColumn>()**

**.Where(c => c.ColumnName != "TickStamp" && c.DataType == typeof(byte)) // Ensure byte params for simplicity**

**.Select(c => c.ColumnName)**

**.OrderBy(name => name) // Sort alphabetically**

**.ToList();**

**foreach (string name in paramNames)**

**{**

**cmbXAxis.Items.Add(name);**

**cmbYAxis.Items.Add(name);**

**cmbZAxis.Items.Add(name);**

**if (cmbTimePlotParamFace != null) cmbTimePlotParamFace.Items.Add(name);**

**if (cmbOutlierParamFace != null) cmbOutlierParamFace.Items.Add(name);**

**}**

**// Restore selections or set defaults**

**SetComboBoxSelection(cmbXAxis, oldX, 0);**

**SetComboBoxSelection(cmbYAxis, oldY, Math.Min(1, cmbYAxis.Items.Count - 1));**

**SetComboBoxSelection(cmbZAxis, oldZ, Math.Min(2, cmbZAxis.Items.Count - 1));**

**if (cmbTimePlotParamFace != null) SetComboBoxSelection(cmbTimePlotParamFace, oldTimeFace, 0);**

**if (cmbOutlierParamFace != null) SetComboBoxSelection(cmbOutlierParamFace, oldOutlierFace, 0);**

**}**

**}**

**private void SetComboBoxSelection(ComboBox cmb, string valueToSet, int defaultIndexIfNotFound)**

**{**

**if (cmb == null || cmb.Items.Count == 0) return;**

**int index = -1;**

**if (!string.IsNullOrEmpty(valueToSet))**

**{**

**index = cmb.FindStringExact(valueToSet);**

**}**

**if (index != -1)**

**{**

**cmb.SelectedIndex = index;**

**}**

**else if (defaultIndexIfNotFound >= 0 && cmb.Items.Count > defaultIndexIfNotFound)**

**{**

**cmb.SelectedIndex = defaultIndexIfNotFound;**

**}**

**else if (cmb.Items.Count > 0) // Fallback to first item**

**{**

**cmb.SelectedIndex = 0;**

**}**

**}**

**// Consolidated event handler for ComboBoxes that trigger data/texture refresh**

**private void cmbMainOrFaceParams\_Changed(object sender, EventArgs e)**

**{**

**PrepareDataAndTextures();**

**}**

**private void chkShowCube\_CheckedChanged(object sender, EventArgs e)**

**{**

**sharpGLControl1.Invalidate(); // Just redraw, no data change**

**}**

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

**{**

**if (saveFileDialog1.ShowDialog() == DialogResult.OK)**

**{**

**try**

**{**

**Bitmap bmp = new Bitmap(sharpGLControl1.Width, sharpGLControl1.Height);**

**OpenGL gl = sharpGLControl1.OpenGL;**

**// Ensure the scene is drawn before reading pixels**

**sharpGLControl1.DoRender(); // Force a render pass if using RenderTrigger.Manual or if needed**

**// Read pixels from the front buffer**

**gl.ReadBuffer(OpenGL.GL\_FRONT);**

**BitmapData bmpData = bmp.LockBits(new Rectangle(0, 0, bmp.Width, bmp.Height),**

**ImageLockMode.WriteOnly,**

**PixelFormat.Format24bppRgb); // Or Format32bppArgb if alpha is needed**

**gl.ReadPixels(0, 0, bmp.Width, bmp.Height,**

**OpenGL.GL\_BGR, // GDI+ uses BGR byte order**

**OpenGL.GL\_UNSIGNED\_BYTE,**

**bmpData.Scan0);**

**bmp.UnlockBits(bmpData);**

**// OpenGL origin is bottom-left, GDI+ is top-left**

**bmp.RotateFlip(RotateFlipType.RotateNoneFlipY);**

**bmp.Save(saveFileDialog1.FileName);**

**MessageBox.Show("Image saved to " + saveFileDialog1.FileName, "Image Saved", MessageBoxButtons.OK, MessageBoxIcon.Information);**

**bmp.Dispose();**

**}**

**catch (Exception ex)**

**{**

**MessageBox.Show($"Error saving image: {ex.Message}", "Save Error", MessageBoxButtons.OK, MessageBoxIcon.Error);**

**}**

**}**

**}**

**}**

**}**