МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ



Кафедра СМАРТ технологий

Лабораторная работа № 2

«Применение проекций и матриц трансляции при визуализации данных»

По дисциплине «Технологии визуализации данных систем управления»

Группа: 221-327

Студент: Шурова Д. С.

Дата: 13.05.2025

Преподаватель: Идиатуллов Т. Т.

Москва, 2025

**Задача**

* Разработать приложение по генерации стохастических данных с заданным профилем распределения и визуализации распределения случайных величин.
* 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].
* Разработать генератор случайных величин на основе пространственной параметрической функции (от номера точки) с добавлением случайного смещения по трем координатам, распределенного по нормальному закону с заданным параметром σ (СКО) [функция определяется индивидуальным заданием к лабораторной работе].
* Разработать систему отображения данных в виде облака точек на битовой карте (bitmap) с использованием прямой (ортогональной) проекции на координатные плоскости. Размер битовой карты – не менее 600 х 600 px. Применять изменение яркости точек для отображения удаленности точек от наблюдателя.
* Разработать систему визуализации с использованием перспективной проекции облака точек, полагая, что область распределения точек можно принять за куб с длиной грани в 1 м, размещенной на расстоянии 4 метров от наблюдателя.
* Разработать систему поворота базовой системы координат размещения облака точек (относительно плоскости экрана) с помощью матрицы трансляции поворота. Реализовать инструмент редактирования матрицы трансляции поворота через DataGrid или текстовые поля.
* Реализовать инструмент инкрементального поворота базовой системы координат путем использования механизма Drag&Drop позиционным манипулятором. Каждое перетаскивание интерпретировать как локальное приращение угла поворота вокруг осей 0X и 0Y в системе координат, связанной с плоскостью экрана, но с центром, размещенным в начале системы координат набора точек.
* Реализовать отображение точек с учетом накопленного поворота, а также отображение базовых векторов системы координат.
* Реализовать сохранение полученного изображения в файл.

**Ход работы**

Метод buttonLoad\_Click реализует загрузку точек из CSV файла, содержащего строки формата (X;Y;Z). Метод btnGenerateParametric\_Click генерирует два набора точек: pointCloud – равномерное распределённые по X, Y, Z в диапазоне [-1, 1]; pointCloudSigma – на основе параметрической функции с добавлением нормального шума. (Рисунок 1)

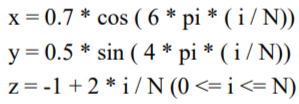


Рисунок 1 – Параметрическая функция

В методе DrawSigmaCloud() происходит отрисовка прямой проекции в pictureBox1, где 2D визуализация. Координаты Z отвечает за яркость цвета. (Рисунок 2)

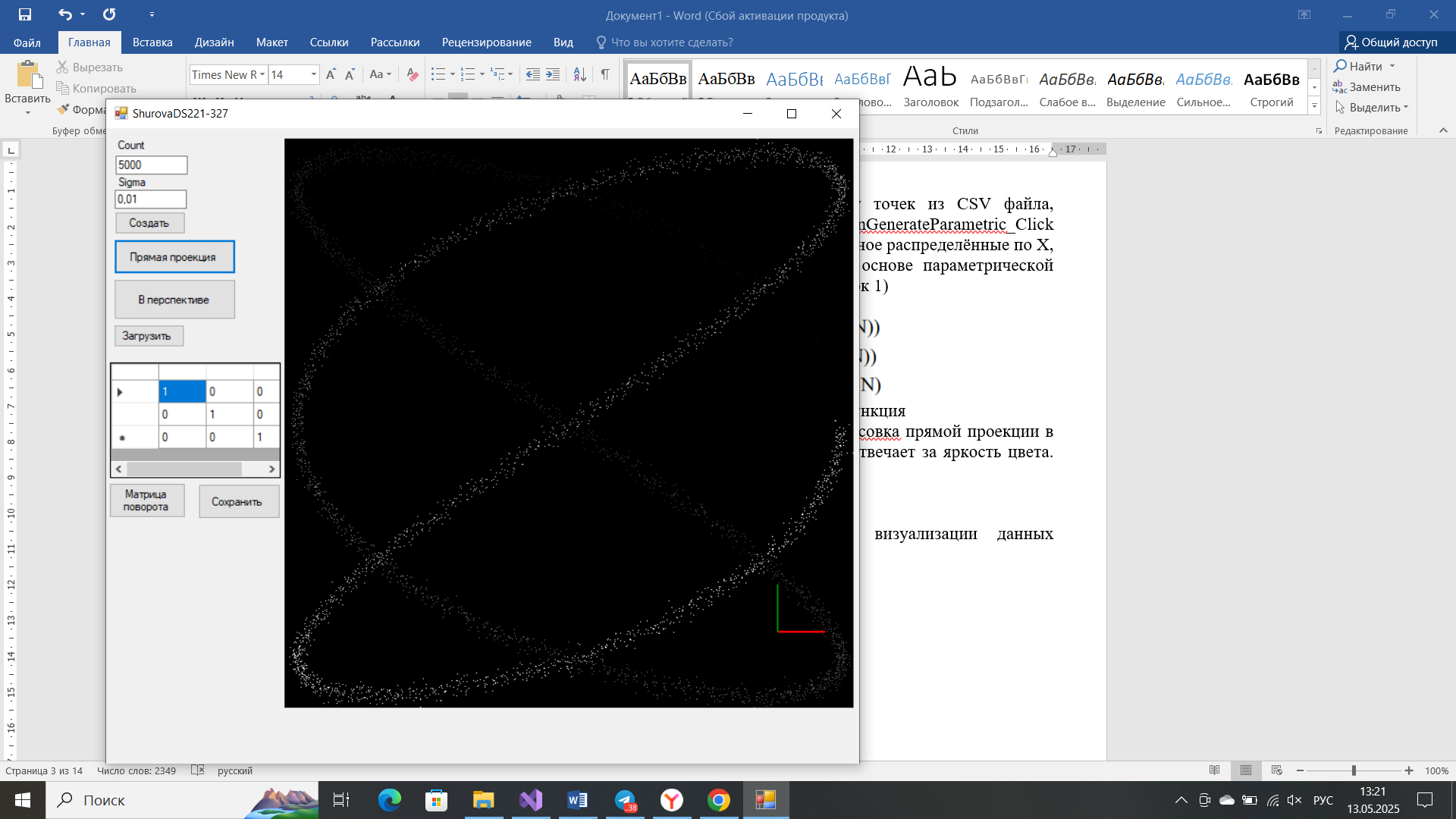


Рисунок 2 – Прямая проекция

Перспективная проекция (на расстояние 4 метров) реализуется в методе DrawPerspectiveProjection(). Используется матрица перспективной проекции 4\*4. Проецируются точки предварительно смещенные в доль оси Z на 4 метра. (Рисунок 3)

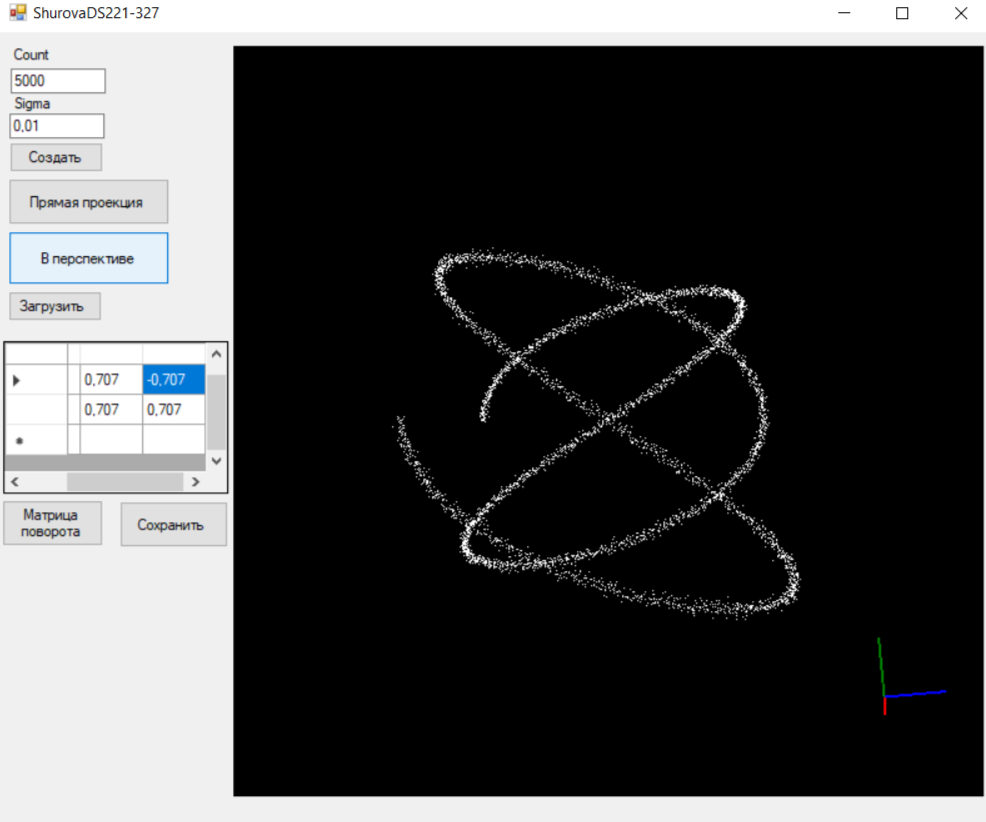


Рисунок 3 – Перспективная проекция отображения точек

Реализовано редактирование матрицы поворота вручную на форме пользователем, матрица 3\*3 отображается в dataGridView1. После применения матрицы все координаты точек пересчитываются. (Рисунок 4)

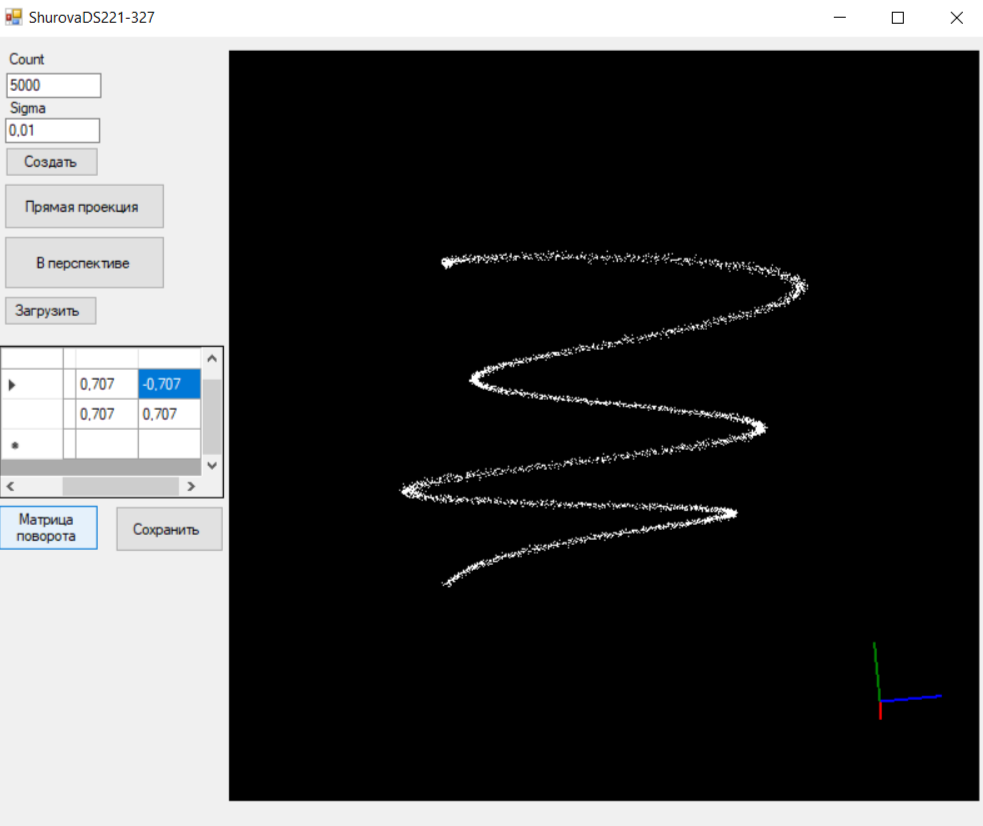
****

Рисунок 4 – Матрица поворота

Вращение системы координат мышью (Drag&Drop) реализовано через события и метод пересчета матрицы. Определяется смещение мыши, вычисляются углы поворота вокруг осей X и Y. Формируются матрицы поворота для X и Y. Применяется поворот к массивам точек, в зависимости от проекции, которая отображается в данный момент.

Реализовано Отображение базисных векторов в методах отображения проекция. Цвета: Красный (X), зеленый (Y), синий (Z). Поворот осей синхронизируется с вращением точек через матрицы.

Происходит сохранение bitmap в файл (путь указан в коде), как картинки png.

**Вывод**

Реализована полноценная система визуализации 3D-точек с возможностью выбора типа проекции. Пользователь может управлять поворотами как вручную, так и через мышь. Введены базисные векторы, синхронизированные с системой координат. Реализовано масштабирование, проецирование и сохранение изображения.

**Приложение А**

Листинг 1 – Код программы

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Windows.Forms;

using static System.Windows.Forms.AxHost;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.ProgressBar;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.Rebar;

namespace Lab2\_VD

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

InitializeRotationMatrixUI();

}

private List<(double X, double Y, double Z)> pointCloud = new List<(double X, double Y, double Z)>();

private List<(double X, double Y, double Z)> pointCloudSigma = new List<(double X, double Y, double Z)>();

private double[,] accumulatedRotationCloud = new double[3, 3]

{

{ 1, 0, 0 },

{ 0, 1, 0 },

{ 0, 0, 1 }

};

private double[,] accumulatedRotationSigma = new double[3, 3]

{

{ 1, 0, 0 },

{ 0, 1, 0 },

{ 0, 0, 1 }

};

private void buttonLoad\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "CSV files (\*.csv)|\*.csv"

};

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

var points = new List<(double X, double Y, double Z)>();

foreach (var line in File.ReadAllLines(openFileDialog.FileName))

{

var parts = line.Split(';');

if (parts.Length == 3 &&

double.TryParse(parts[0], out double x) &&

double.TryParse(parts[1], out double y) &&

double.TryParse(parts[2], out double z))

{

points.Add((x, y, z));

}

}

pointCloud = points;

}

}

int N = 0;

double sigma = 0;

private void ParametricSigma ()

{

pointCloudSigma.Clear();

Random rand = new Random();

// Второй список: параметрическая функция + нормальное смещение

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

{

double t = (double)i / N;

// Параметрическая функция

double x = 0.7 \* Math.Cos(6 \* Math.PI \* t);

double y = 0.5 \* Math.Sin(4 \* Math.PI \* t);

double z = -1 + 2 \* t;

// Генерация смещений по нормальному закону

double dx = GenerateGaussian(rand, 0, sigma);

double dy = GenerateGaussian(rand, 0, sigma);

double dz = GenerateGaussian(rand, 0, sigma);

pointCloudSigma.Add((x + dx, y + dy, z + dz));

}

// Масштабируем pointCloudSigma в интервал [-1, 1]

pointCloudSigma = ScaleToMinusOneToOne(pointCloudSigma);

}

private void btnGenerateParametric\_Click(object sender, EventArgs e)

{

if (!int.TryParse(txtCount.Text, out N) || N <= 0)

{

MessageBox.Show("Введите корректное положительное число точек.");

return;

}

if (!double.TryParse(textSigma.Text, out sigma) || sigma < 0)

{

MessageBox.Show("Введите корректное значение σ (СКО).");

return;

}

pointCloud.Clear();

pointCloudSigma.Clear();

Random rand = new Random();

// Первый список: равномерно распределенные точки

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

{

double x = rand.NextDouble() \* 2 - 1;

double y = rand.NextDouble() \* 2 - 1;

double z = rand.NextDouble() \* 2 - 1;

pointCloud.Add((x, y, z));

}

ParametricSigma();

}

// Метод генерации смещения по нормальному закону с помощью Box-Muller

private double GenerateGaussian(Random rand, double mean, double stddev)

{

double u1 = 1.0 - rand.NextDouble(); // [0,1) -> (0,1]

double u2 = 1.0 - rand.NextDouble();

double randStdNormal = Math.Sqrt(-2.0 \* Math.Log(u1)) \* Math.Cos(2.0 \* Math.PI \* u2);

return mean + stddev \* randStdNormal;

}

// Масштабирование координат по каждой оси в интервал [-1, 1]

private List<(double X, double Y, double Z)> ScaleToMinusOneToOne(List<(double X, double Y, double Z)> points)

{

double minX = points.Min(p => p.X);

double maxX = points.Max(p => p.X);

double minY = points.Min(p => p.Y);

double maxY = points.Max(p => p.Y);

double minZ = points.Min(p => p.Z);

double maxZ = points.Max(p => p.Z);

return points.Select(p => (

X: Scale(p.X, minX, maxX),

Y: Scale(p.Y, minY, maxY),

Z: Scale(p.Z, minZ, maxZ)

)).ToList();

}

// Преобразование значения в интервал [-1, 1]

private double Scale(double value, double min, double max)

{

if (max == min) return 0; // защита от деления на 0

return -1 + 2 \* (value - min) / (max - min);

}

private void DrawSigmaCloud()

{

if (pointCloudSigma == null || pointCloudSigma.Count == 0)

{

MessageBox.Show("Сначала сгенерируйте точки.");

return;

}

int size = 600; // Размер битовой карты

Bitmap bmp = new Bitmap(size, size);

Graphics g = Graphics.FromImage(bmp); // <== добавлено!

g.Clear(Color.Black);

double minX = pointCloudSigma.Min(p => p.X);

double maxX = pointCloudSigma.Max(p => p.X);

double minY = pointCloudSigma.Min(p => p.Y);

double maxY = pointCloudSigma.Max(p => p.Y);

double minZ = pointCloudSigma.Min(p => p.Z);

double maxZ = pointCloudSigma.Max(p => p.Z);

foreach (var p in pointCloudSigma)

{

// Прямая проекция на плоскость XY

int x = (int)((p.X - minX) / (maxX - minX) \* (size - 1));

int y = (int)((p.Y - minY) / (maxY - minY) \* (size - 1));

y = size - 1 - y; // инверсия по оси Y для корректного отображения

// Яркость зависит от Z: чем ближе точка (больше Z), тем ярче

int brightness = 255;

if (maxZ > minZ)

brightness = (int)(255 \* (p.Z - minZ) / (maxZ - minZ));

Color color = Color.FromArgb(brightness, brightness, brightness);

if (x >= 0 && x < size && y >= 0 && y < size)

{

bmp.SetPixel(x, y, color);

}

}

DrawBasisVectors(g, size); // <== теперь работает

pictureBox1.Image = bmp;

}

private void btnDrawCloud\_Click(object sender, EventArgs e)

{

proekcia = true;

DrawSigmaCloud();

}

public struct Vector3

{

public double X, Y, Z;

public Vector3(double x, double y, double z)

{

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

}

public double[] ToHomogeneous()

{

return new double[] { X, Y, Z, 1.0 };

}

}

private void btnDrawPerspective\_Click(object sender, EventArgs e)

{

proekcia = false;

DrawPerspectiveProjection();

}

private void DrawPerspectiveProjection()

{

if (pointCloud == null || pointCloud.Count == 0)

{

MessageBox.Show("Сначала сгенерируйте точки.");

return;

}

int size = 600;

Bitmap bmp = new Bitmap(size, size);

Graphics g = Graphics.FromImage(bmp);

g.Clear(Color.Black);

// Параметры перспективной проекции

double d = 4.0; // расстояние до куба

double fov = Math.PI / 3; // угол обзора (60 градусов)

double aspect = 1.0;

double near = 1.0;

double far = 10.0;

// Построение перспективной матрицы

double f = 1.0 / Math.Tan(fov / 2);

double[,] projectionMatrix = new double[4, 4]

{

{ f / aspect, 0, 0, 0 },

{ 0, f, 0, 0 },

{ 0, 0, (far + near) / (near - far), (2 \* far \* near) / (near - far) },

{ 0, 0, -1, 0 }

};

foreach (var point in pointCloud)

{

// Смещение куба вдоль оси Z на расстояние d от наблюдателя

double x = point.X;

double y = point.Y;

double z = point.Z + d;

// Вектор в однородных координатах

double[] vec = new double[] { x, y, z, 1.0 };

// Умножение на матрицу проекции

double[] projected = new double[4];

for (int i = 0; i < 4; i++)

{

projected[i] = 0;

for (int j = 0; j < 4; j++)

projected[i] += projectionMatrix[i, j] \* vec[j];

}

// Перевод из однородных координат

if (projected[3] == 0) continue;

double px = projected[0] / projected[3];

double py = projected[1] / projected[3];

// Приведение к координатам на Bitmap (от -1 до 1 -> 0 до size)

int sx = (int)((px + 1) / 2 \* (size - 1));

int sy = (int)((1 - (py + 1) / 2) \* (size - 1)); // инверсия Y

if (sx >= 0 && sx < size && sy >= 0 && sy < size)

{

bmp.SetPixel(sx, sy, Color.White);

}

}

DrawBasisVectors(g, size);

pictureBox1.Image = bmp;

}

private double[,] currentRotationMatrix = new double[3, 3];

private void InitializeRotationMatrixUI()

{

dataGridViewRotation.ColumnCount = 3;

dataGridViewRotation.RowCount = 3;

for (int i = 0; i < 3; i++)

{

dataGridViewRotation.Columns[i].Width = 50;

for (int j = 0; j < 3; j++)

{

dataGridViewRotation[i, j].Value = i == j ? 1.0 : 0.0;

currentRotationMatrix[i, j] = (i == j) ? 1.0 : 0.0;

}

}

}

private void UpdateMatrixFromGrid()

{

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

currentRotationMatrix[i, j] = Convert.ToDouble(dataGridViewRotation[j, i].Value);

}

private void btnMatrix\_Click(object sender, EventArgs e)

{

ApplyRotationFromGrid();

}

private void ApplyRotationFromGrid()

{

if (pointCloud == null || pointCloud.Count == 0)

{

MessageBox.Show("Сначала загрузите или сгенерируйте точки.");

return;

}

// Считываем матрицу поворота из DataGridView

UpdateMatrixFromGrid();

var rotatedPoints = new List<(double X, double Y, double Z)>();

foreach (var p in pointCloud)

{

double x = currentRotationMatrix[0, 0] \* p.X + currentRotationMatrix[0, 1] \* p.Y + currentRotationMatrix[0, 2] \* p.Z;

double y = currentRotationMatrix[1, 0] \* p.X + currentRotationMatrix[1, 1] \* p.Y + currentRotationMatrix[1, 2] \* p.Z;

double z = currentRotationMatrix[2, 0] \* p.X + currentRotationMatrix[2, 1] \* p.Y + currentRotationMatrix[2, 2] \* p.Z;

rotatedPoints.Add((x, y, z));

}

pointCloud = rotatedPoints;

// Визуализируем повернутые точки в перспективной проекции

DrawPerspectiveProjection();

}

private bool isDragging = false;

private Point lastMousePosition;

private bool proekcia = false;

private void pictureBox1\_MouseDown(object sender, MouseEventArgs e)

{

isDragging = true;

lastMousePosition = e.Location;

}

private void pictureBox1\_MouseMove(object sender, MouseEventArgs e)

{

if (!isDragging) return;

int dx = e.X - lastMousePosition.X;

int dy = e.Y - lastMousePosition.Y;

// Преобразуем пиксели в градусы

double angleX = dy \* 0.5; // Вокруг оси X — движение по Y

double angleY = dx \* 0.5; // Вокруг оси Y — движение по X

// В радианы

double radX = angleX \* Math.PI / 180.0;

double radY = angleY \* Math.PI / 180.0;

// Построим комбинированную матрицу поворота: сначала по X, потом по Y

double[,] rotationX =

{

{1, 0, 0},

{0, Math.Cos(radX), -Math.Sin(radX)},

{0, Math.Sin(radX), Math.Cos(radX)}

};

double[,] rotationY =

{

{Math.Cos(radY), 0, Math.Sin(radY)},

{0, 1, 0},

{-Math.Sin(radY), 0, Math.Cos(radY)}

};

lastMousePosition = e.Location;

if (proekcia)

{

// Применим последовательно X, затем Y

ApplyRotation(rotationX, pointCloudSigma);

ApplyRotation(rotationY, pointCloudSigma);

DrawSigmaCloud();

}

else {

// Применим последовательно X, затем Y

ApplyRotation(rotationX, pointCloud);

ApplyRotation(rotationY, pointCloud);

DrawPerspectiveProjection(); } // перерисовать

}

private void pictureBox1\_MouseUp(object sender, MouseEventArgs e)

{

isDragging = false;

}

private void ApplyRotation(double[,] matrix, List<(double X, double Y, double Z)> point)

{

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

{

var p = point[i];

double x = p.X, y = p.Y, z = p.Z;

double newX = matrix[0, 0] \* x + matrix[0, 1] \* y + matrix[0, 2] \* z;

double newY = matrix[1, 0] \* x + matrix[1, 1] \* y + matrix[1, 2] \* z;

double newZ = matrix[2, 0] \* x + matrix[2, 1] \* y + matrix[2, 2] \* z;

if (proekcia) { pointCloudSigma[i] = (newX, newY, newZ); }

else { pointCloud[i] = (newX, newY, newZ); ; }

}

if (proekcia)

accumulatedRotationSigma = MultiplyMatrices(matrix, accumulatedRotationSigma);

else

accumulatedRotationCloud = MultiplyMatrices(matrix, accumulatedRotationCloud);

}

private double[,] MultiplyMatrices(double[,] A, double[,] B)

{

double[,] result = new double[3, 3];

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

for (int k = 0; k < 3; k++)

result[i, j] += A[i, k] \* B[k, j];

return result;

}

private void DrawBasisVectors(Graphics g, int size)

{

Point origin = new Point(size - 80, size - 80); // начало системы координат в углу

double scale = 50; // длина стрелки в пикселях

var matrix = proekcia? accumulatedRotationSigma : accumulatedRotationCloud;

(double dx, double dy) Transform(double x, double y, double z)

{

double tx = matrix[0, 0] \* x + matrix[0, 1] \* y + matrix[0, 2] \* z;

double ty = matrix[1, 0] \* x + matrix[1, 1] \* y + matrix[1, 2] \* z;

return (tx, -ty);

}

using (Pen penX = new Pen(Color.Red, 2))

using (Pen penY = new Pen(Color.Green, 2))

using (Pen penZ = new Pen(Color.Blue, 2))

{

var (dxX, dyX) = Transform(1, 0, 0);

var (dxY, dyY) = Transform(0, 1, 0);

var (dxZ, dyZ) = Transform(0, 0, 1);

g.DrawLine(penX, origin, new PointF(origin.X + (float)(dxX \* scale), origin.Y + (float)(dyX \* scale)));

g.DrawLine(penY, origin, new PointF(origin.X + (float)(dxY \* scale), origin.Y + (float)(dyY \* scale)));

g.DrawLine(penZ, origin, new PointF(origin.X + (float)(dxZ \* scale), origin.Y + (float)(dyZ \* scale)));

}

}

private void SaveBtn\_Click(object sender, EventArgs e)

{

// Проверяем, есть ли изображение в pictureBox1

if (pictureBox1.Image == null)

{

MessageBox.Show("Нет изображения для сохранения!", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

// Указываем путь для сохранения (измените на свой)

string savePath = @"C:\Users\LENOVO\source\repos\Lab2\_VD\Image.png";

// Сохраняем изображение

pictureBox1.Image.Save(savePath, System.Drawing.Imaging.ImageFormat.Png);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении изображения: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}